DOBOT
Share Creating Pleasure

TEXHONAB
EXAMEN TECHNOLAB.RU

DOBOT MAGICIAN®

ОБРАЗОВАТЕЛЬНАЯ ИНЖЕНЕРНАЯ ПЛАТФОРМА

ПРОГРАММИРОВАНИЕ МАНИПУЛЯТОРА В СРЕДЕ PYTHON

О. А. Горнов

```
Rb = 0
Rg = 0
Rk = 0
while True:
    dType.SetEMotorEx(api, 0, 1, 10000, 1)
    if (dType.GetInfraredSensor(api, 1)[0]) == 1:
        dType.dSleep(200)
        dType.SetEMotorEx(api, 0, 0, 0, 1)
        dType.SetPTPCmdEx(api, 0, 279,  78,  4, 0, 1)
        dType.SetEndEffectorSuctionCupEx(api, 1, 1)
        dType.dSleep(
        pos = dType.GetPose(api)
```

DOBOT.RU
DOBOT.EXAMEN-TECHNOLAB.RU

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Как устроены цифровые производства? Весь процесс разбивается на ряд задач. Есть задачи манипуляций и локомоций (перемещений). С ними мы познакомились в прошлой книге. А есть задачи обработки материалов, им будет посвящена данная книга. Вся обработка материалов сводится к двум противоположным принципам. Первый — что-то скрепить или напечатать слоями, а потом скрепить, — это аддитивные технологии. Второй — наоборот, убрать из детали лишнее — это субтрактивные технологии. Так работает, например, фреза или лазер. Обычно для того, чтобы выполнить такие задачи, используется связка из графического редактора, в котором пользователь осуществляет чертеж (инвентор — система трехмерного проектирования, предназначенная для создания цифровых прототипов промышленных изделий, компас, блендер), и редактора, который переведет чертеж в коды, понятные станку, так называемые G-коды. И для той и для другой программы ключевым действием является построение поверхностей. Этими поверхностями являются графические объекты. Поэтому первой задачей нашего пособия мы считаем описание и обучение построению геометрических объектов. Вторая цель — научить использовать язык программирования Python для программирования робота Dobot Magician. Язык программирования Python является универсальным языком программирования: он позволяет быстро писать и быстро отлаживать код, он очень популярен, для него есть большое количество библиотек, в том числе и библиотеки для технического зрения, и самое важное для нас — библиотека для Dobot. Кроме того, как было показано в предыдущей книге, код, который мы пишем в Blockly, дублируется и в Python. Так что Dobot — подходящее решение для изучения языка Python.

ЗАДАНИЕ I. Настройка Dobot для программирования на языке Python. Настройка рабочего инструмента

Для того чтобы работа с Dobot была наиболее удобной и быстрой, необходимо подключить его к управляющему компьютеру при помощи сети ***Wi-Fi***. Для этого необходимо использовать модуль ***Wireless-2***.

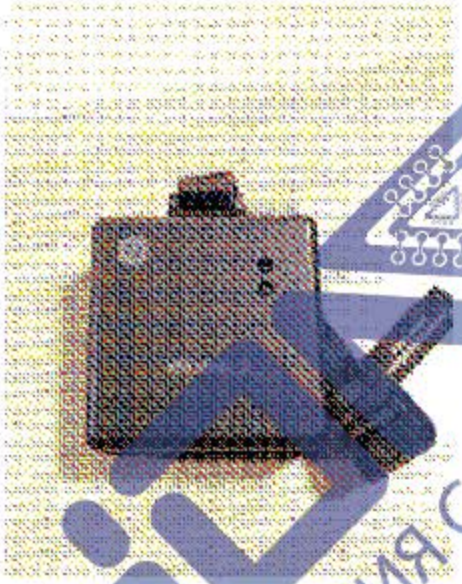


Рисунок I-1. Модуль Wi-Fi связи Wireless-2

Данный модуль подключается в гнездо Communication Interface в задней части Dobot.

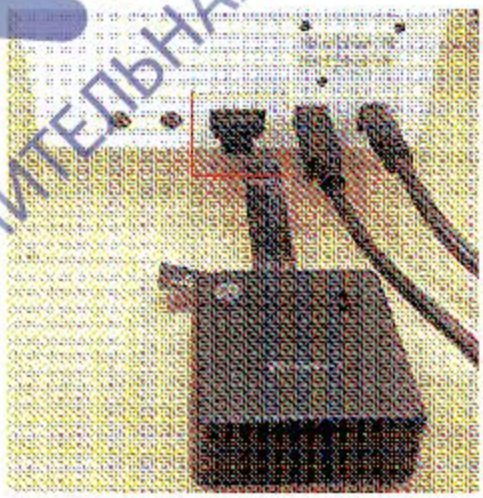


Рисунок I-2. Гнездо Communication Interface

Затем необходимо включить Dobot Magician и дождаться двух коротких звуковых сигналов. При этом включится синий светодиодный индикатор.

Для дальнейшей работы нам потребуется среда DobotStudio, которую можно бесплатно скачать на сайте dobot.cc в меню ***Support*** подменю ***Download Center***.

Подключите Dobot к компьютеру через кабель USB. Зайдите в меню ***Настройки (Settings)*** в правом верхнем углу рабочего экрана.

Рисунок I-3. Расположение меню ***Настройки***

Выберите пункт меню ***Wi-Fi***. В этом пункте заполните поля ***SSID*** — имя вашей сети ***Wi-Fi***. В поле ***Пароль (Password)*** введите пароль сети. Остальные поля должны заполниться автоматически. Если этого не произойдет, вам необходимо будет самостоятельно их настроить. Для этого следует зайти в настройки сети на вашем компьютере и скопировать параметры.

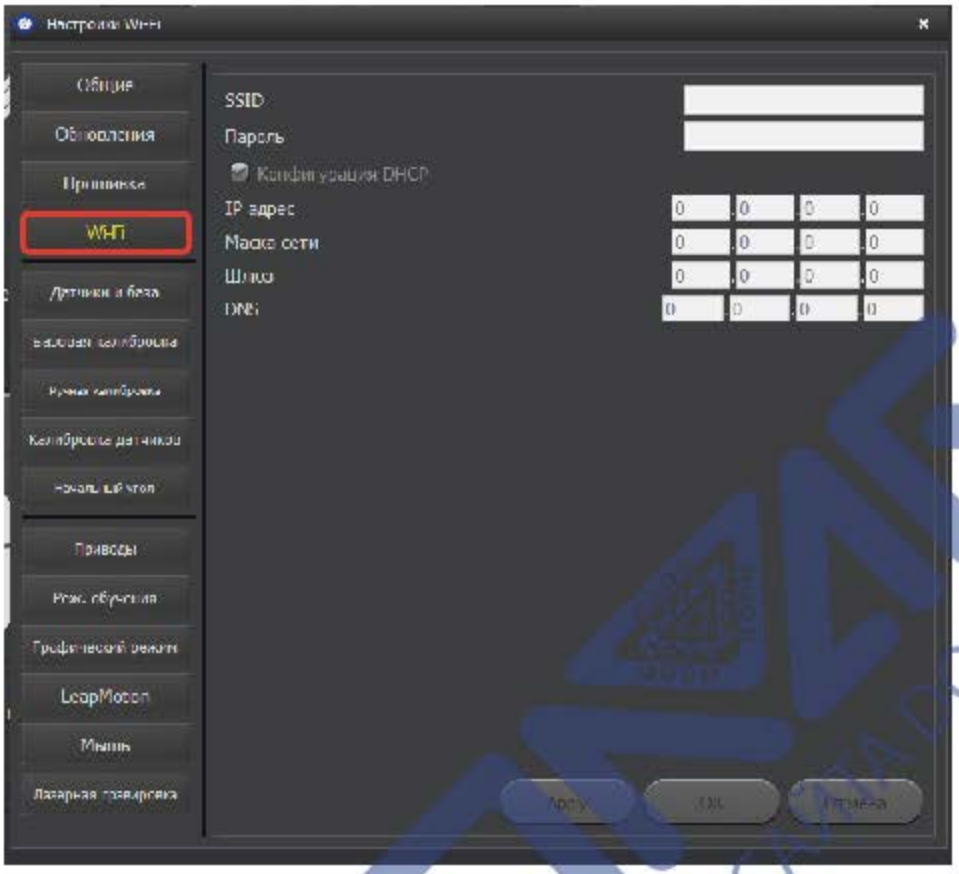


Рисунок I-4. Окно настройки Wi-Fi

В случае если Dobot установил соединение на модуле ***Wi-Fi***-связи, зажжется зеленый светодиод — модуль Wireless-2 готов к работе.

Рисунок I-5. Модуль Wireless-2 готов к работе

Подождите 5 секунд. Нажмите кнопку ***Отключить (Disconnect)*** в левом верхнем углу экрана. Обратите внимание на то, что для доступных подключений появился IP-адрес. Выберите это подключение. Нажмите ***Подключить (Connect)*** — Dobot готов к работе.

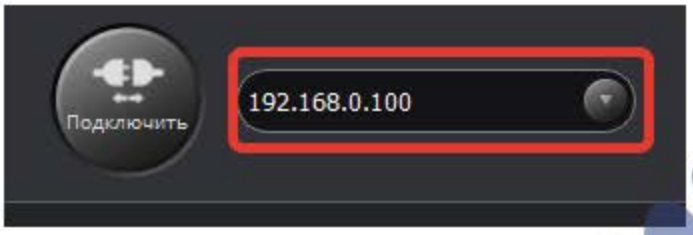


Рисунок I-6. Кнопка Подключения/Отключения Dobot

В этом курсе программирование поведения Dobot будет осуществляться на языке программирования Python. Для того чтобы переход со среды программирования Blockly осуществлялся как можно проще, разработчики манипулятора реализовали окно ***Основной код (General cod)***, которое находится справа от рабочего поля программы Blockly.

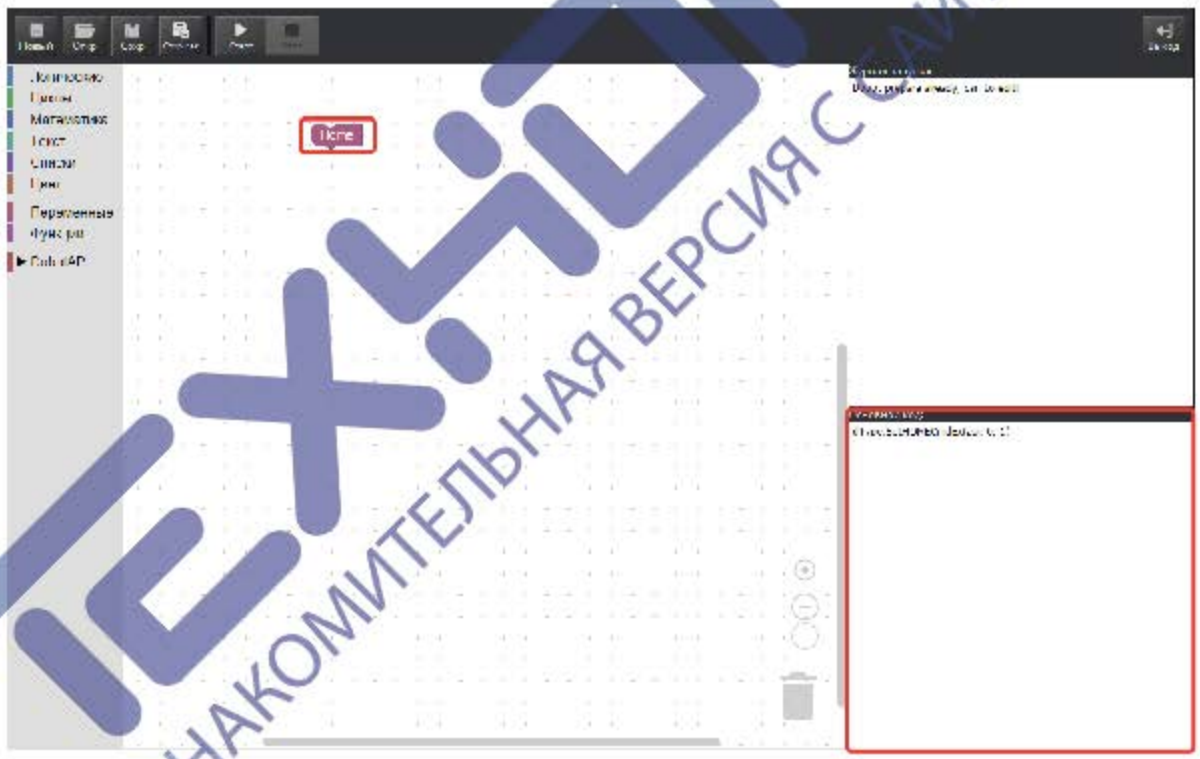


Рисунок I-7. Расположение окна ***Основной код***

Обратите внимание, что требуется выполнить только одну операцию ***Home***. Эта операция позволяет Dobot найти начальную точку положения рабочего инструмента. Для этого он отклоняется по часовой стрелке в самое крайнее положение, а затем возвращается к координатам $x = 260, y = 0, z = -8$.

Рисунок I-8. Координаты начальной точки рабочего инструмента Dobot

Для начала изучения библиотек Python при работе с Dobot достаточно будет скопировать текст из окна *Основной код* в *Режим Script*.

Рисунок I-9. Режим Script

Данный режим предназначен для того, чтобы писать код на языке Python. После написания кода Dobot осуществит все команды. Для того чтобы запустить код, необходимо нажать на кнопку ***Старт (Start)*** в верхней части экрана.

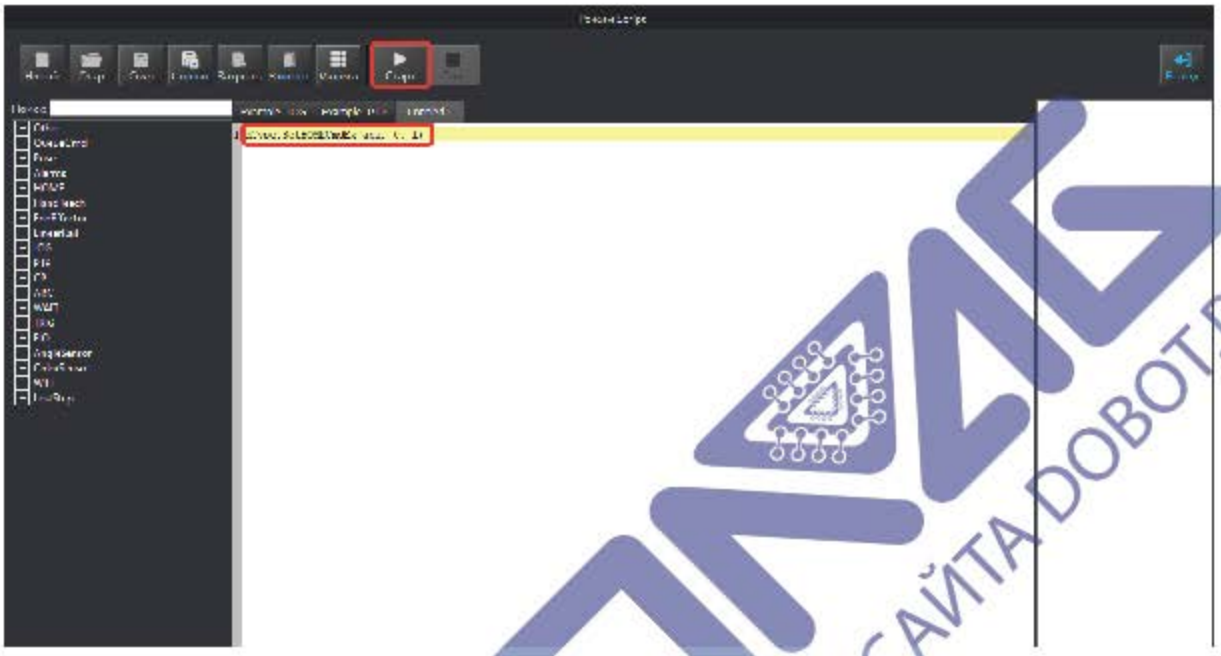


Рисунок I-10. Запуск кода Python

Dobot выполнит операцию ***Home***.

Кроме того, существует возможность писать код непосредственно в Python. Для этого необходимо на сайте dobot.cc перейти во вкладку ***Support***, выбрать пункт ***Download Center***, открыть папку Dobot Magician и скачать архив ***Dobot Demo v2.0***, находящийся внутри папки ***Development protocol***.

В этом архиве есть подробное описание, как использовать различные языки, среды программирования и платформы для управления Dobot Magician. Нам же необходимо перенести все файлы из папки ***DobotDemoForPython*** в папку, где находится язык программирования ***Python*** на вашем компьютере.

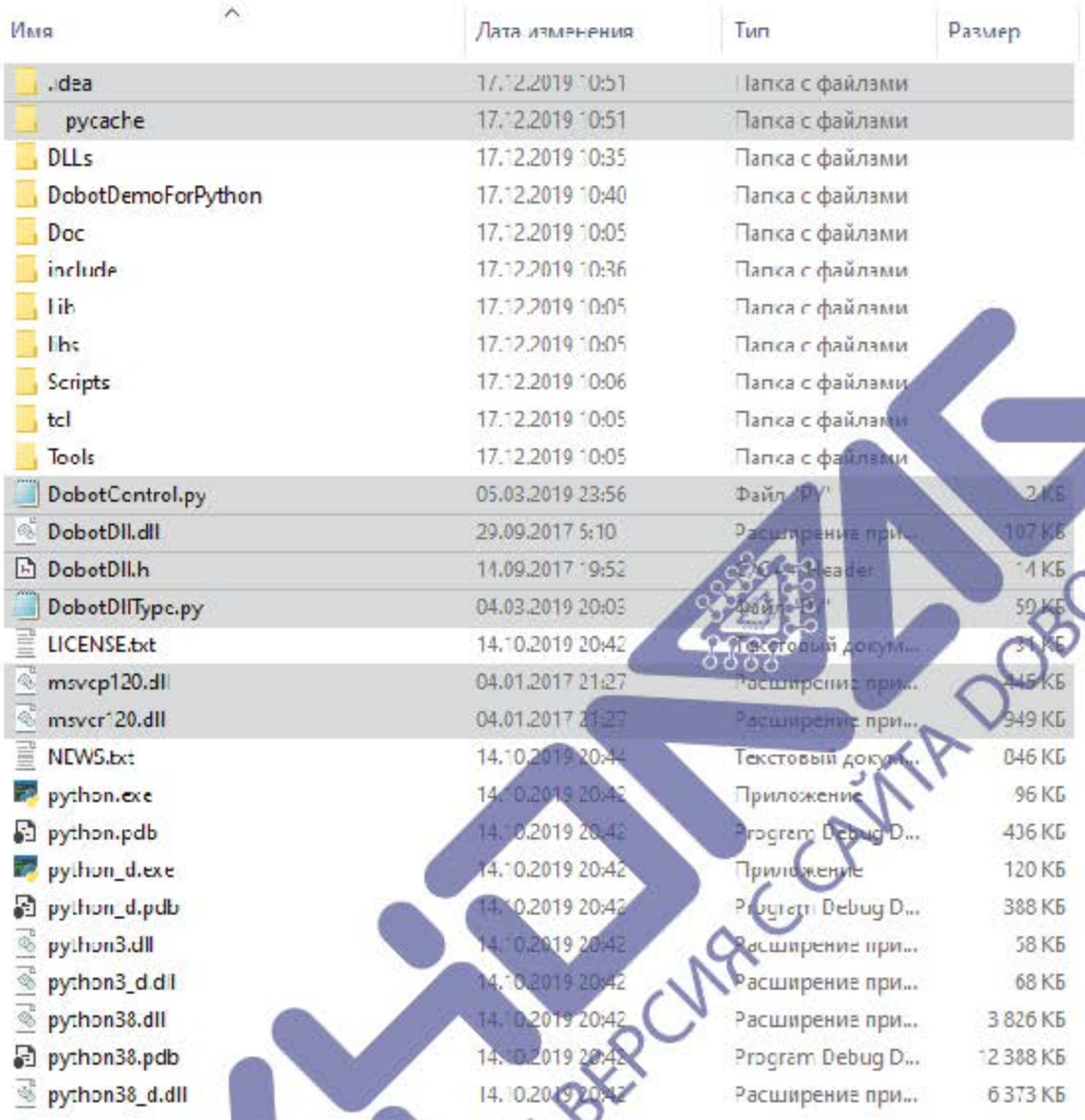


Рисунок I-11. Подготовка к работе в языке программирования Python

Откроем в среде ***IDLE*** файл ***DobotControl.py***. Код этого файла осуществляет подключение всех необходимых библиотек Dobot, а также необходимые команды по его подключению и проверке подключения. Давайте вставим код для выполнения команды ***Home***:

dType.SetHOMECmdEx(api, 0, 1)

в конец программы:

```
            offset = 50
        else:
            offset = -50
        lastIndex = dType.SetPTPCmd(api, dType.PTPMode.PTPMOVLXYZMode,

    # This  EMotor
    # This  EMotor
    # This  EMotor
    # This  EMotor

    # dType.SetEMotor(api, 0, 1, 10000, isQueued=1)
    # dType.SetEMotorS(api, 0, 1, 10000, 20000,isQueued=1)


    #Start to Execute Command Queued
    dType.SetQueuedCmdStartExec(api)

    #Wait for Executing Last Command
    while lastIndex > dType.GetQueuedCmdCurrentIndex(api)[0]:
        dType.dSleep(100)

    #Stop to Execute Command Queued
    dType.SetQueuedCmdStopExec(api)

#Disconnect Dobot
dType.DisconnectDobot(api)
dType.SetHOMECmdEx(api, 0, 1)
```

*Рисунок I-12. Применение команды **Home** в среде IDLE языка программирования Python*

При запуске данного кода придет сообщение о том, что Dobot подключен к компьютеру и осуществлена операция ***Home***. Такой подход может быть удобен, если есть необходимость использовать все возможности языка Python и его многочисленных библиотек, таких как библиотеки машинного зрения. Часть заданий в этом пособии будет реализована именно в этой среде.

Обратите внимание: при использовании Dobot вне среды DobotStudio невозможно осуществить его коммуникацию через Wi-Fi-интерфейс. Подключение необходимо проводить через USB-кабель.

Последняя настройка, которую необходимо произвести перед началом работы, это настройка высоты инструмента в нулевом положении. Для этого нам понадобится **Auto-levelling tool** — инструмент автоматического выравнивания.

Рисунок I-13. Auto-levelling tool — инструмент автоматического выравнивания

Прикрепите его в место крепления рабочего инструмента, как показано на рисунке ниже, и затяните винт.

Рисунок I-14. Инструмент автоматического выравнивания в гнезде рабочего инструмента Dobot

Подключите кабель инструмента автоматического выравнивания в гнездо ***GP4***, как показано на рисунке I-15.

Рисунок I-15. Подключение кабеля инструмента автоматического выравнивания в гнездо GP4

Зайдите в меню ***Настройки*** и выберите пункт меню ***Калибровка датчиков (Auto Leveling)***.

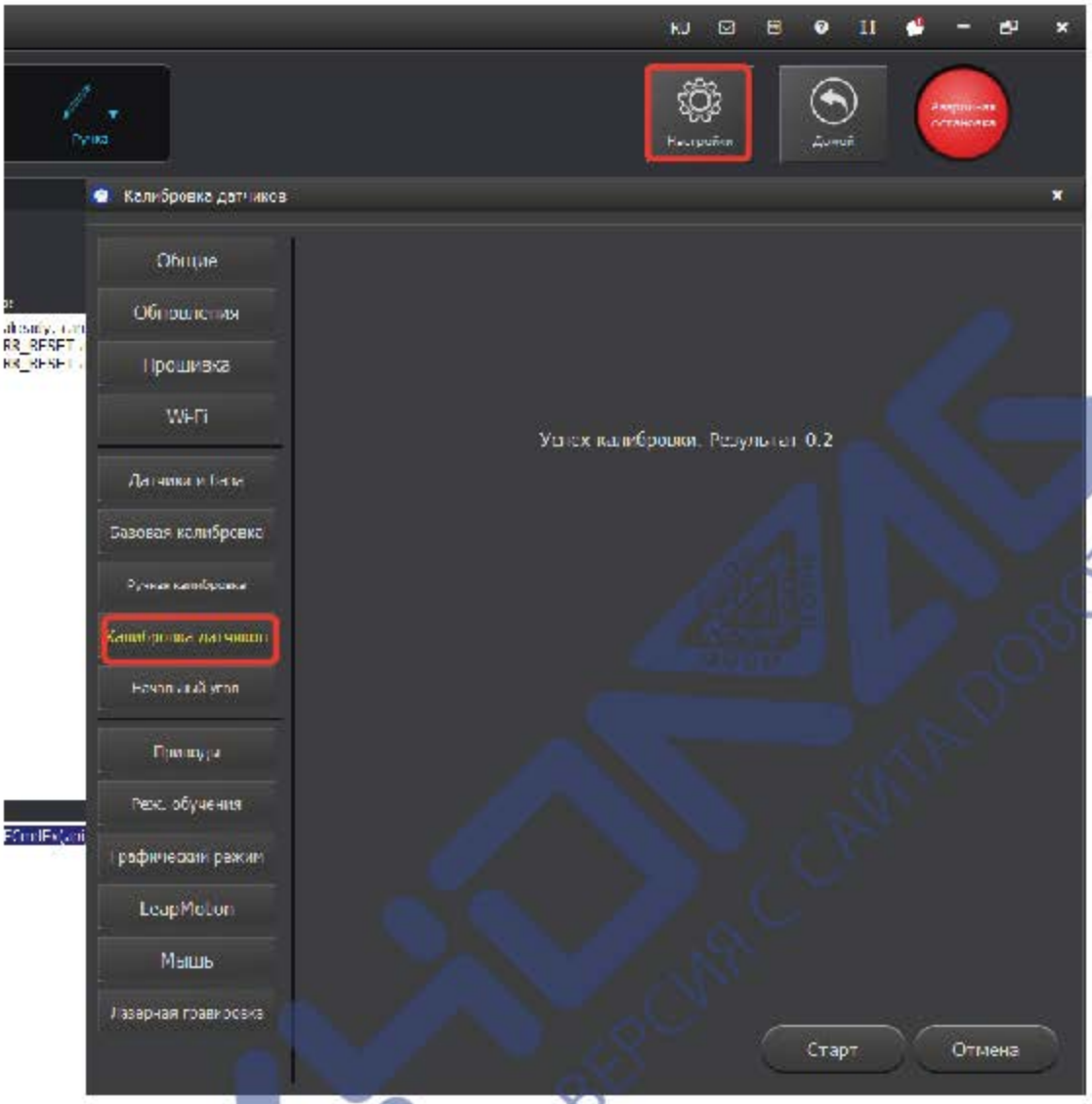


Рисунок I-16. Пункт меню ***Настройки: Калибровка датчиков***

Калибровка продлится некоторое время, это может занять около одной минуты.

Теперь, когда все настройки произведены, приступаем к установке и настройке рабочего инструмента. В целом это пособие рассказывает, как создать программы для получения геометрических примитивов с использованием в качестве рабочего инструмента лазера, но тот же самый код с небольшим видоизменением можно использовать и для рисования ручкой. Такое решение будет более безопасным и, возможно, более востребованным на уроках.

Задания для самостоятельной работы:

1) осуществить настройку рабочих инструментов Dobot;
2) составить несколько простых программ в среде Blockly, соотнести полученный код с программами, записанными в окне **Основной код**;
3) реализовать программы из предыдущей книги в среде Python в режиме **Script**.

ЗАДАНИЕ II. Особенности языка Python. Реализация базовых команд Dobot на языке программирования Python. Реализация простых графических объектов: точка, прямая, прямоугольник

Прежде всего необходимо настроить лазер для работы. Нам может понадобиться несколько режимов. Первый — когда лазер используется только как инструмент рисования. Второй режим — это режим резания. Кроме того, понадобится и комбинация указанных выше режимов, когда часть фигуры должна быть вырезана, а часть сложена определенным образом. В любом случае все режимы работы лазера зависят от таких факторов, как:

1) плотность обрабатываемого материала;
2) толщина материала;
3) цвет поверхности материла;
4) расстояние от лазера до обрабатываемой поверхности;
5) продолжительность воздействия лазера на одну точку;
6) фокусировка лазера на поверхность.

Последние три фактора могут регулироваться как программно (4, 5), так и за счет настроек окуляра (6).

Установим лазер в качестве рабочего инструмента. Для этого используйте фиксирующий винт, который обозначен на рисунке цифрой 1.

Рисунок II-1. Установка лазера в качестве рабочего инструмента Dobot

Подключите кабели лазера к портам GP5 и SW4.

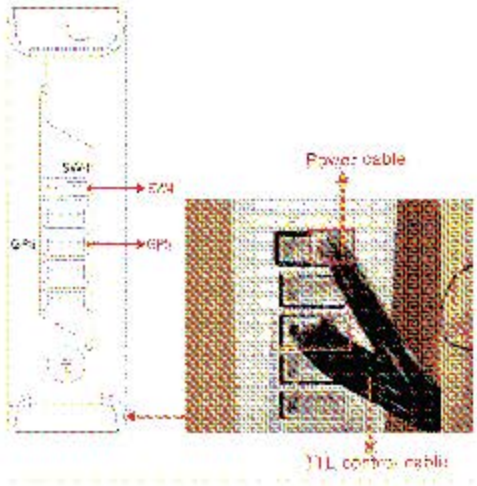


Рисунок II-2. Подключение кабелей рабочего инструмента (лазера) к манипулятору Dobot

В верхней части окна программы DobotStudio выберите Лазер (Laser) в качестве рабочего инструмента.

Рисунок II-3 Выбор лазера в качестве рабочего инструмента

Лазер готов к работе. Для того чтобы убедиться в этом, используйте стойкую к возгоранию поверхность в качестве подложки, на которой будет находиться обьект, предназначенный для обработки. Для этого подойдет лист стекла или металла. Важно, чтобы подложку не мог прожечь лазер. В качестве обрабатываемого материала лучше всего использовать плотный картон, но подойдут и листы канцелярской бумаги.

Для включения лазера поставьте галочку в окно Лазер (Laser) на панели управления в правой части окна программы DobotStudio. Для выключения лазера необходимо убрать галочку из этого окна.

*Рисунок II-4. Окно **Лазер (Laser)** на панели управления **DobotStudio***

После включения осуществите фокусировку луча лазера на обрабатываемой поверхности. Для этого необходимо поворачивать окуляр лазера, как показано на рисунке, до тех пор пока световой след от лазера на поверхности не станет минимального размера.

ВНИМАНИЕ! ПРИ ЛЮБЫХ ДЕЙСТВИЯХ С ЛАЗЕРОМ НЕОБХОДИМО ИСПОЛЬЗОВАТЬ ЗАЩИТНЫЕ ОЧКИ. В ПРОТИВНОМ СЛУЧАЕ ВЕЛИКА ВЕРОЯТНОСТЬ ОЖОГА СЕТЧАТКИ ГЛАЗА.

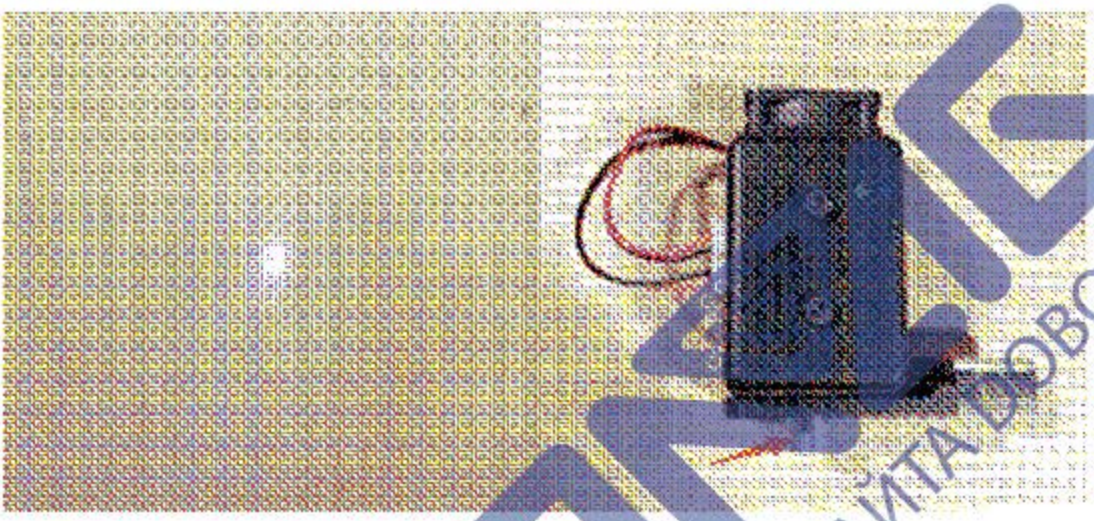


Рисунок II-5. Фокусировка лазера на обрабатываемой поверхности

Настало время осуществить первое кодирование работы Dobot в Python. Для того чтобы лучше понять код, осуществим часть его в среде Blockly.

Задание: необходимо выполнить операцию ***Home***, а затем выжечь точку с координатами $x = 300$; $y = 0$.

Код в Blockly будет выглядеть следующим образом:

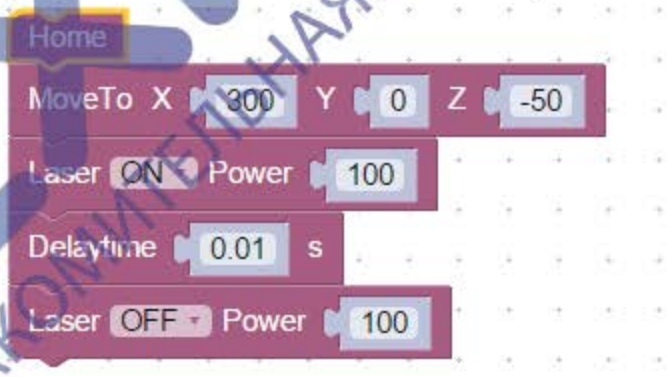


Рисунок II-6. Выжигание одной точки с координатами $x = 300$; $y = 0$

Для выжигания точки на листе канцелярской бумаги необходима высота инструмента $z = -50$ и время 0,01 с. По окончании этого времени лазер выключается.

Теперь перенесем код из окна **Основной код** в **Режим Script**. Запустим программу нажатием на клавишу **Старт** (**Start**).

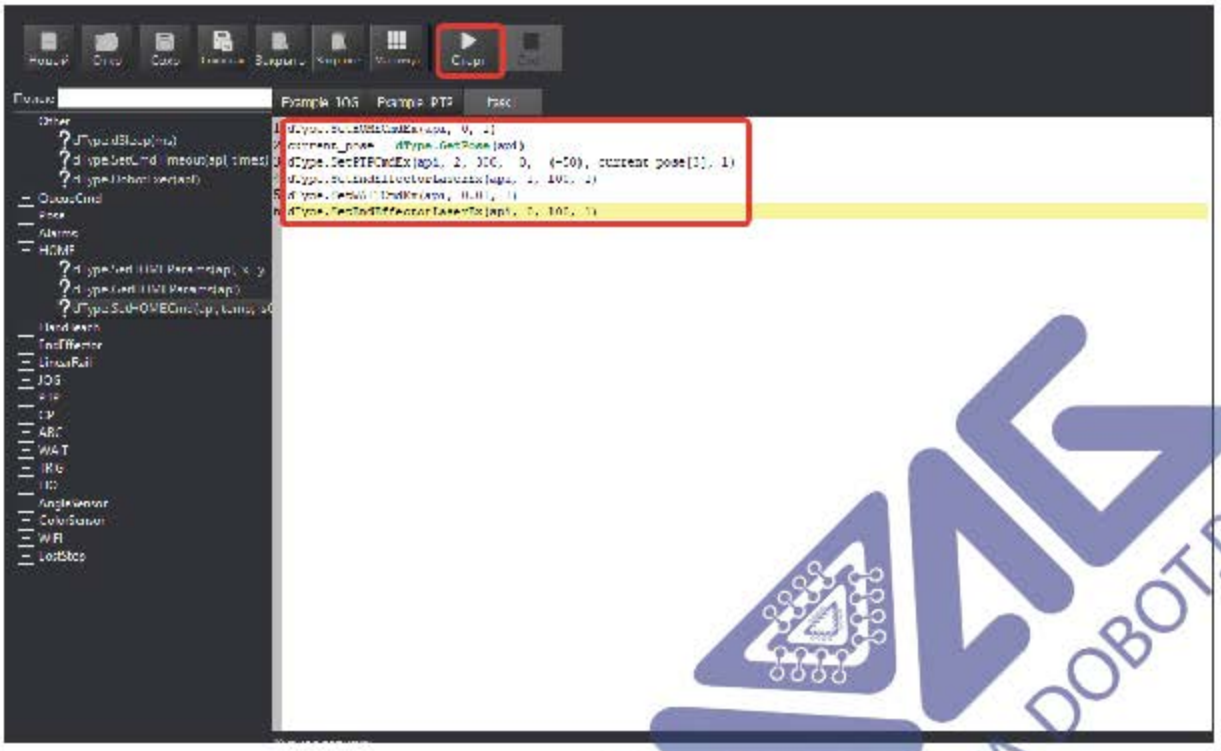


Рисунок II-7. Выжигание одной точки с координатами $x = 300, y = 0$ в Python

Обратите внимание на то, что Dobot выполняет те же действия, что и в среде Blockly.

Код в Python выполняется сверху вниз слева направо. Для того чтобы интерпретатор различал команды, их необходимо размещать в разных строках или разделять точкой с запятой в одной строке. Важной особенностью кода в Python являются отступы. Отступы объединяют группу команд, которые выполняются единым блоком, например команды, составляющие тело одного цикла. О таких конструкциях речь пойдет в следующих главах. Сейчас же важно отметить, что весь код, который выполняется последовательно, должен быть написан без отступов.

Настало время понять назначение и аргументы функций, используемых в Python для управления Dobot. Разработчики манипулятора создали для программирования в этом языке программирования специальную библиотеку, которая содержит небольшое количество функций методов, с назначением которых надо разобраться. Для упрощения этой задачи в левой части окна программы DobotStudio в **Режиме Script** расположены все специфические для Dobot функции и методы.

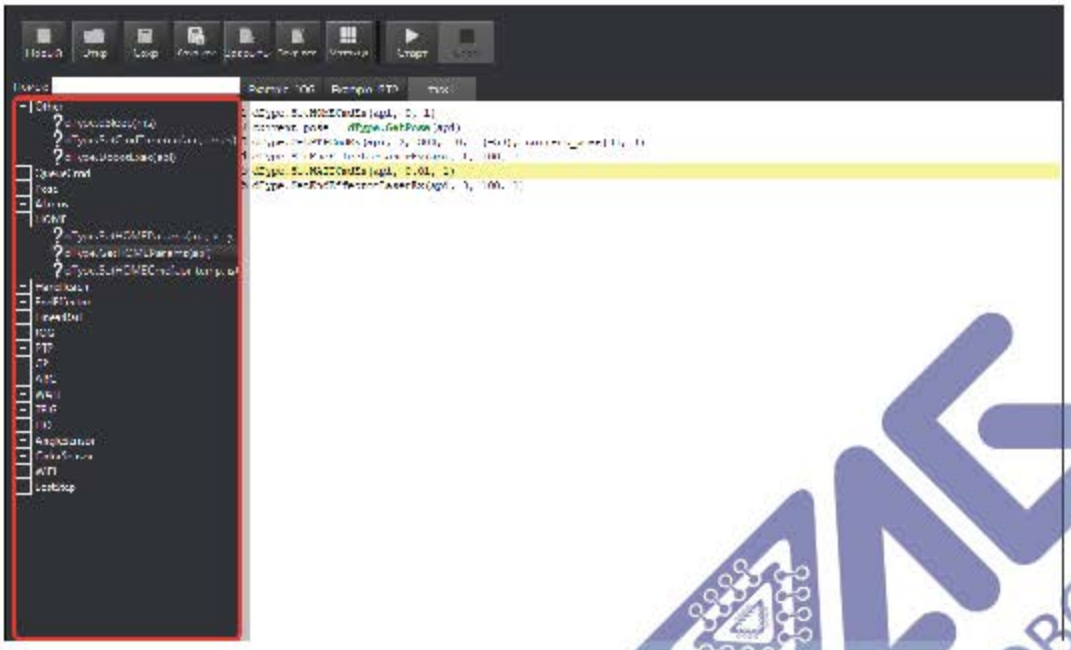


Рисунок II-8. Список специфических для Dobot функций языка Python

При нажатии на вопросительный знак рядом с функцией или методом можно получить справку о них. Например, для функции

dType.SetHOMECmd(api, 0, 1)

расписаны все ее аргументы и возвращаемое значение.

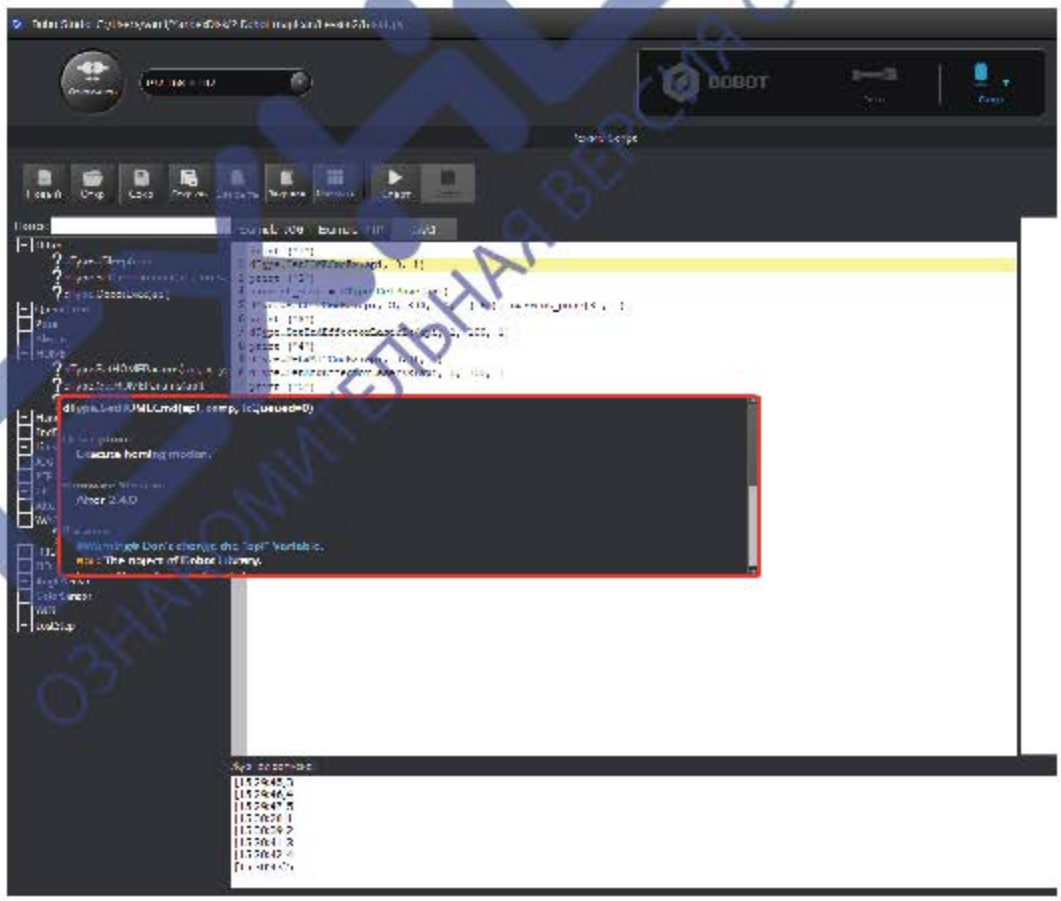


Рисунок II-9. Подсказка (Help) по функции ***dType.SetHOMECmd***

Так, первый аргумент **api** — это обьект Dobot Librari, его изменять нельзя, второй аргумент **temp** — это значение нам не понадобится. Третий же аргумент является переключателем режима работы Dobot — **the switch state of Queue mode**. Изменять этот режим нет необходимости, но может понадобиться изменить начальную точку, в которую возвращается рабочий инструмент при выполнении действия **Home**. Для этого необходимо воспользоваться функцией **dType.SetHOMEParams(api, x, y, z, r, isQueued=0)**. Ее аргументы, помимо перечисленных выше, это ***x***, ***y***, ***z*** — координаты начальной точки. Изменяйте их с осторожностью, поскольку точка обязательно должна находиться в рабочей зоне. Обратите внимание и на способ обращения к функции библиотеки Dobot. Сначала идет обращение к самой библиотеке ***dType,*** затем ставится точка, а за ней название функции. Можно вызывать функции ***dType*** и другим способом, но об этом будет рассказано в следующих главах.

Вернемся к нашему коду, выжигающему одну точку. Во второй строчке на рисунке расположена функция

dType.GetPose(api).

Эта функция возвращает текущие координаты рабочего инструмента. В этой же строчке происходит присваивание этих координат списку ***current_pose***. Список — это особый вид данных в языке Python, в данном случае ***current_pose*** содержит координаты рабочего инструмента в декартовой системе координат, а также положение валов шаговых двигателей и сервопривода рабочего инструмента в градусах. Для того чтобы считать и вывести на экран координату *x*, необходимо выполнить следующий код:

print (***current_pose[0]***).

Для ***y*** этот код будет выглядеть несколько иначе:

print (***current_pose[1]***).

Таким образом, можно сделать вывод, что функция ***dType.GetPose(api)*** возвращает не одно значение, а целый список. Подробнее о назначении каждого элемента можно узнать, нажав на знак вопроса.

В строке номер три происходит перемещение рабочего инструмента. Для этого используется функция:

dType.SetPTPCmd(api, 2, 300, 0, (-50), current_pose[3], 1).

В функции интересными для нас являются аргументы 2, 3, 4, 5, 6. Аргумент номер 2 характеризует тип перемещения, подробнее о них рассказывалось в предыдущей книге. Аргументы 3, 4, 5 — это координаты рабочего инструмента ***x***, ***y***, ***z***, а аргумент 6 — это угол поворота рабочего инструмента сервопривода. В нашем коде при перемещении он остается неизменным и считывается из списка ***current_pose***. Важно заметить, что все эти элементы являются целыми числами.

После перемещения осуществляется включение лазера

dType.SetEndEffectorLaserEx(api, 1, 100, 1).

Второй аргумент этой функции — это 1 или 0 — лазер включен или выключен соответственно. Третий аргумент — мощность лазера.

Следующая строка — это пауза в 0,01 секунды, о чем говорит второй аргумент функции.

В последней строке лазер выключается. Мы не будем впредь столь подробно разбирать аргументы каждой из используемых функций, поскольку вы можете самостоятельно реализовать код в среде Blockly, скопировать его в Python из окна **Основной код** (**General code**), а затем реализовать его в режим ***Script***, изучив при этом все необходимые аргументы и функции, нажимая на вопросительный знак рядом с каждой функцией.

Для каждой задачи размер и очертания рабочей зоны могут быть своими, но, для того чтобы понять их границы, проще всего перейти во вкладку **Write&Draw** (рисунок II-10). Самая большая площадь рабочей области будет в том случае, если координата ***z*** при любых перемещениях будет равна 0. В этом случае манипулятор может «вытягиваться» на максимальное расстояние по оси ***x***. Необходимо заметить, что при работе с ***лазером*** в качестве рабочего инструмента можно выбрать почти любое значение ***z***, так как при помощи окуляра лазер настраивается на любую высоту.

Рисунок II-10. Минимальный и максимальный радиусы доступа рабочего инструмента Dobot

В меню **Write&Draw** на панели управления выставьте значение для оси ***z***, равное 0. Изменяя положение ***x*** рабочего инструмента, найдите максимальное и минимальное значения, характеризующие его положение в крайних точках — дальней и ближней от основания соответственно. Запишите эти значения. В нашем случае они равны:

$$R_{MAX} = 340,$$
$$R_{MIN} = 122.$$

Важно отметить, что при значении x = 340 невозможно перемещение ни по оси z, ни по оси y. Так, например, если необходимо выжечь или нарисовать наибольшую прямую, параллельную оси x, этот режим будет предпочтительным, но в большинстве случаев нам будет необходимо найти некоторое сбалансированное значение между шириной и длиной рабочей области, как это показано на рисунке ниже. Возможна и прямоугольная область печати — в этом случае длина будет больше или меньше ширины.

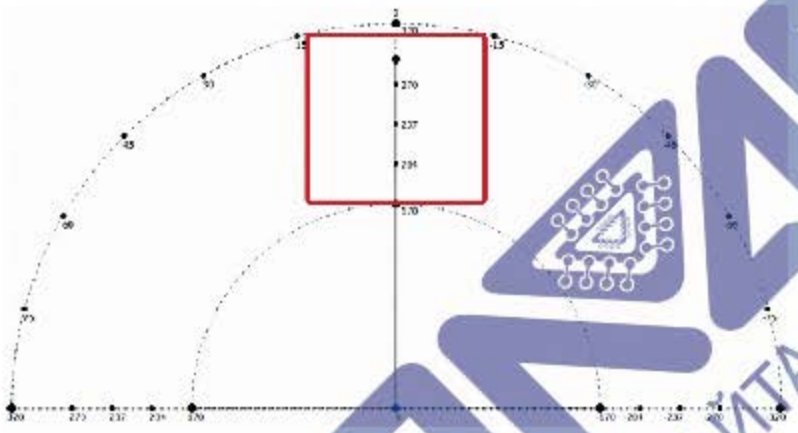


Рисунок II-11. Квадратная область доступа рабочего инструмента Dobot

Давайте рассчитаем максимальную область печати при заданной длине или ширине. Для этого рассмотрим следующее построение:

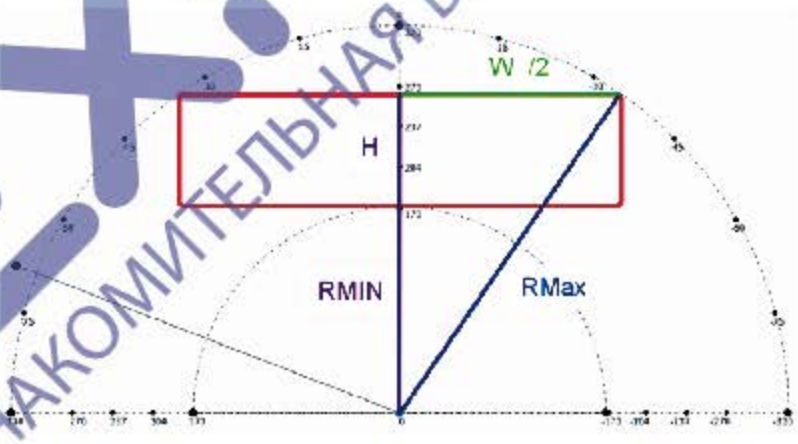


Рисунок II-12. Соотношение длины и ширины прямоугольной рабочей области

Ширина и длина рабочей области — это W и H. R_{MIN} и R_{MAX} — это радиусы, которые характеризуют предельные значения отклонения рабочего инструмента от опоры. Запишем для полученного прямоугольного треугольника теорему Пифагора:

$$(R_{MAX})^2 = (R_{MIN} + H)^2 + (W/2)^2.$$

Из этого соотношения можно получить следующее выражение:

$$(W/2)^2 = (R_{MAX})^2 - (R_{MIN} + H)^2$$

Используем формулу для разности квадратов:

$$(W/2)^2 = (R_{MAX} - R_{MIN} - H) \cdot (R_{MAX} + R_{MIN} + H).$$

Окончательная формула будет иметь вид:

$$W = 2((R_{MAX} - R_{MIN} - H) \cdot (R_{MAX} + R_{MIN} + H))^{1/2}.$$

Такую формулу можно использовать перед началом использования Dobot. В случае если перемещения будут осуществляться только внутри рабочей зоны, манипулятор будет работать бесперебойно. Эту формулу можно вставить в табличный редактор, например Excel, и получить примерные соотношения длины и ширины. На рисунке П-13 представлены таблица и график функции зависимости длины от ширины для робота, который мы использовали при написании книги.

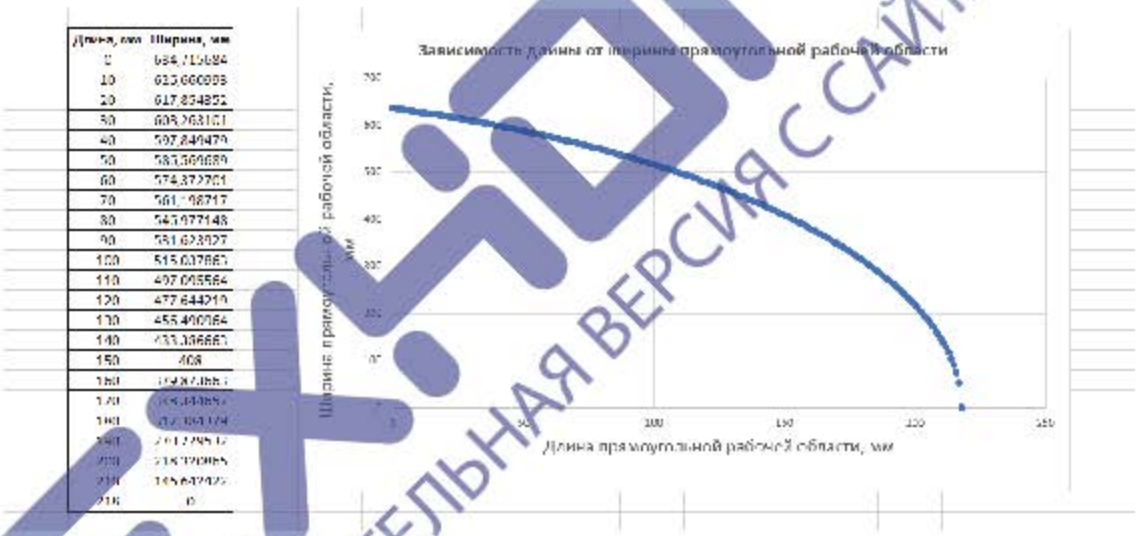


Рисунок II-13. Соотношение длины и ширины прямогольной рабочей зоны Dobot

Вы можете получить свои результаты, однако они мало будут отличаться от представленных ниже. Давайте проанализируем полученные графическую зависимость и таблицу. При длине рабочей зоны в 0 миллиметров получается самая большая ширина. При уменьшении ширины рабочей зоны растет ее длина. При высоте в 200 миллиметров получается рабочая зона квадратной формы. Таким образом, получается, что наибольшая рабочая зона квадратной формы — это квадрат со стороной в 200 мм. Осталось рассчитать границы, за которые не должен выходить рабочий инструмент Dobot. Для квадратной области это будет:

$$122 < x < 322,$$
$$-100 < y < 100.$$

Эти значения получены прибавлением к наименьшей границе по оси ***x*** высоты 200 мм, а также вычитанием и прибавлением к ***0*** значения в 100 мм (два раза по 100 получается ширина рабочей зоны). При максимально возможной длине рабочей зоны в 218 мм ее ширина будет равна нулю.

Хорошей привычкой станет использование данной формулы или таблицы перед работой над любым проектом. Имея представление о размере рабочей области, вы никогда не отправите рабочий инструмент в «запрещенную» область, где Dobot не будет нормально функционировать. Кроме того, учтите, что все приведенные выше расчеты справедливы для координаты ***z*** рабочего инструмента в 0 мм. Для всех остальных случаев, например при использовании инструмента ***Ручка*** (***Pen***), это значение будет другим. Вам нужно экспериментальным путем найти лишь минимальный и максимальный радиусы, которые ограничивают перемещение Dobot, а затем вставить эти значения в формулу выше.

В следующем задании необходимо поставить лазером точки в углы квадрата, который ограничивает размеры максимальной рабочей зоны. Для этого надо выжечь четыре точки с координатами ***x*** и ***y*** (100; 322), (–100; 322), (100; 122), (–100; 122).

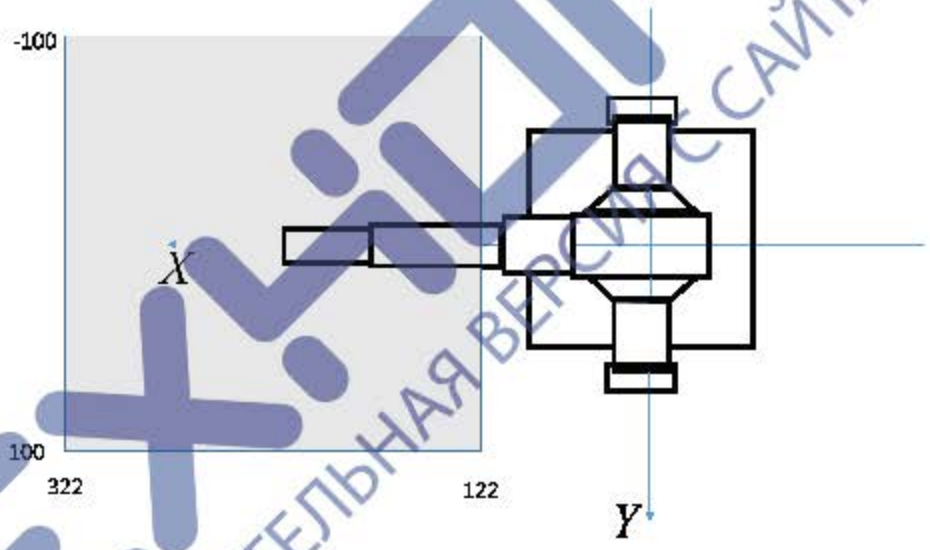


Рисунок II-14

Код для этой задачи будет выглядеть следующим образом:

```
"""Поиск нулевой точки"""

dType.SetHOMECmdEx(api, 0, 1)

"""Текущие координаты записаны в переменную current_pose"""

current_pose = dType.GetPose(api)

"""Перемещение в первую точку и ее выжигание"""
```

```
dType.SetPTPCmdEx(api, 2, 322, 100, 0, current_pose[3], 1)
dType.SetEndEffectorLaserEx(api, 1, 100, 1)
dType.dSleep(10)
dType.SetEndEffectorLaserEx(api, 0, 100, 1)

"""Перемещение во вторую точку и ее выжигание"""

dType.SetPTPCmdEx(api, 2, 322, (-100), 0, current_pose[3], 1)
dType.SetEndEffectorLaserEx(api, 1, 100, 1)
dType.dSleep(10)
dType.SetEndEffectorLaserEx(api, 0, 100, 1)

"""Перемещение в третью точку и ее выжигание"""

dType.SetPTPCmdEx(api, 2, 122, (-100), 0, current_pose[3], 1)
dType.SetEndEffectorLaserEx(api, 1, 100, 1)
dType.dSleep(10)
dType.SetEndEffectorLaserEx(api, 0, 100, 1)

"""Перемещение в четвертую точку и ее выжигание"""

dType.SetPTPCmdEx(api, 2, 122, (100), 0, current_pose[3], 1)
dType.SetEndEffectorLaserEx(api, 1, 100, 1)
dType.dSleep(10)
```

Обратите внимание: мы снабдили программу комментариями внутри самой программы для того, чтобы ей было удобнее пользоваться. Комментарии находятся внутри тройных кавычек. Весь код, помещенный внутрь тройных кавычек, не будет исполняться языком программирования Python. Есть еще один способ комментировать строки — это использование символа #. Весь код, находящийся в той же строке, что и этот символ (справа от него), не будет исполняться программой.

Границы рабочей зоны можно обозначить не только точками, но и прямыми. Для этого необходимо незначительно модернизировать код из предыдущего примера:

```
"""Текущие координаты записаны в переменную current_pose"""

current_pose = dType.GetPose(api)

"""Перемещение в первую точку и ее выжигание"""

dType.SetPTPCmdEx(api, 2, 322, 100, 0, current_pose[3], 1)
dType.SetEndEffectorLaserEx(api, 1, 100, 1)
dType.dSleep(10)

"""Перемещение во вторую точку"""

dType.SetPTPCmdEx(api, 2, 322, (-100), 0, current_pose[3], 1)
dType.dSleep(10)
```

```
"""Перемещение в третью точку"""

dType.SetPTPCmdEx(api, 2, 122, (-100), 0, current_pose[3], 1)
dType.dSleep(10)

"""Перемещение в четвертую точку"""

dType.SetPTPCmdEx(api, 2, 122, (100), 0, current_pose[3], 1)
dType.dSleep(10)
dType.SetEndEffectorLaserEx(api, 0, 100, 1)
```

Из программы было убрано выключение лазера в каждой из точек, за исключением последней. Однако после запуска программы видно, что лазер выжигает только те точки, которые находятся на углах квадрата. Это объясняется тем, что лазер просто не успевает прожечь все точки. Таким образом, для решения этой задачи необходимо еще и замедлить перемещение Dobot при помощи функции

dType.SetPTPCommonParamsEx(api,20,20,1):

```
"""Задается скорость и ускорение Dobot при перемещении от точки к точке"""

dType.SetPTPCommonParamsEx(api,20,20,1)

"""Текущие координаты записаны в переменную current_pose"""

current_pose = dType.GetPose(api)

"""Перемещение в первую точку и ее выжигание"""

dType.SetPTPCmdEx(api, 2, 322, 100, 0, current_pose[3], 1)
dType.SetPTPCommonParamsEx(api,2,2,1)
dType.SetEndEffectorLaserEx(api, 1, 100, 1)
dType.dSleep(10)

"""Перемещение во вторую точку"""

dType.SetPTPCmdEx(api, 2, 322, (-100), 0, current_pose[3], 1)
dType.dSleep(10)

"""Перемещение в третью точку"""

dType.SetPTPCmdEx(api, 2, 122, (-100), 0, current_pose[3], 1)
dType.dSleep(10)

"""Перемещение в четвертую точку"""

dType.SetPTPCmdEx(api, 2, 122, (100), 0, current_pose[3], 1)
dType.dSleep(10)
dType.SetEndEffectorLaserEx(api, 0, 100, 1)
```

Обратите внимание, что функция

dType.SetPTPCommonParamsEx(api,20,20,1)

используется в программе два раза: в первый — для быстрого перемещения к первой точке, во второй — для медленного перемещения между точками, чтобы гарантировать осуществление прожигания контура рабочей области.

Задание для самостоятельной работы: найти размеры рабочих областей, которые могут понадобиться вам для решения собственных задач, и очертить их двумя способами: выжиганием вершин и граней.

ЗАДАНИЕ III. Циклы на языке программирования Python. Точки на линии с различными промежутками. Реализация *for* и *while*

Одним из самых значительных преимуществ использования языков программирования является возможность использовать циклы. Циклы — это многократно повторяющиеся операции. Обычно, как и в языке программирования Python, цикл выполняется до тех пор, пока выполняется какое-либо условие. Например, в следующем случае используется цикл ***while***. В круглых скобках указано условие выполнения.

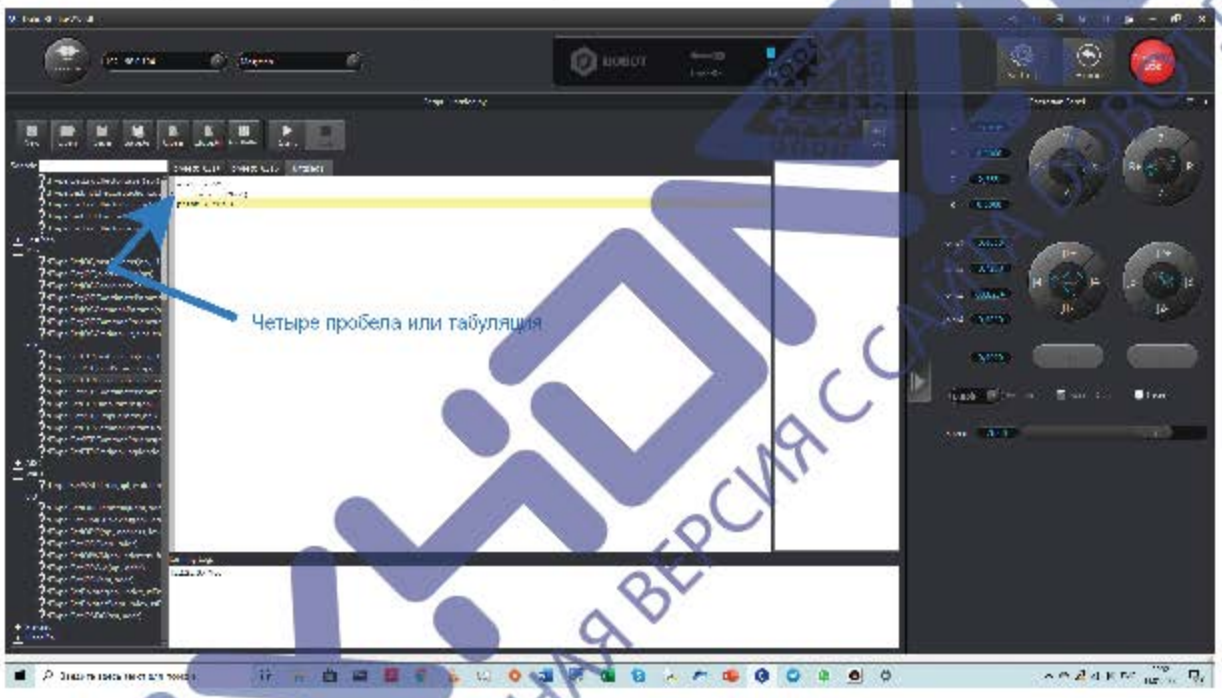


Рисунок III-1. Использование цикла while в Python

Если это условие выполняется, то выполняется ***Тело цикла***. В нашем случае в скобках находится ложное выражение, поэтому тело цикла выполнено не будет. В теле же цикла находится команда ***print***, которая выводит на экран то выражение, которое находится внутри скобок. Если это выражение представляет из себя текст, то его необходимо взять в двойные или одинарные кавычки. Важно отметить, что блок кода, который относится к телу цикла, печатается на расстоянии в 4 пробела или одной табуляции от линии, которая проходит слева от буквы w в слове ***while***.

После того как в нашем коде тело цикла не будет выполнено ни разу, выполняется команда из третьей строке print (“Yes”), о чем свидетельствует запись в ***Журнале запуска (Running Log)***.

Рассмотрим другой пример:

```
1 i = 0
2 while (i < 2):
3     dType.SetPTPCmd(api, 2, 260, 60, 0, 0, isQueued=0)
4     dType.SetPTPCmd(api, 2, 260, -60, 0, 0, isQueued=0)
5     i = i + 1
6
```

Рисунок III-2. Цикл ***while*** *с условием*

В данном примере в строке 1 создается переменная ***i***, которой присваивается значение 0. Таким образом, = это операция присваивания. Прежде чем выполнять тело цикла, программа выполняет проверку условия $i < 2$. По сути, это вопрос с двумя вариантами ответа: ***Истина***, ***Ложь***. В нашем случае это ***Истина***. В 3 и 4 строчке Dobot производит перемещение рабочего инструмента.

В строке 5 используется конструкция:

$$i = i + 1.$$

В этой строке две операции: операция сложения и упоминаемая выше операция присваивания. У всех операций в языках программирования есть свой приоритет — порядок выполнения операций, если они находятся в одном выражении. Это похоже на порядок действий в математических вычислениях. В выражении, где нет скобок, первыми выполняются все операции умножения, и только затем сложения. В данном случае первой выполняется операция сложения, то есть $i + 1$ — это 1 (до этой операции i была равна 0), а затем операция присваивания — и i становится 1.

Так как блок команд, относящихся к телу цикла, выполнен, снова осуществляется проверка условия выполнения цикла во второй строке. В нашем случае i равно 1, что меньше 2. Цикл — перемещение инструмента можно выполнить снова.

После второго выполнения цикла переменная i будет равна 2. При проверке условия выполнения выражение $2 < 2$ окажется ложным, и цикл выполнен не будет.

Однократное выполнение всех команд и операций внутри цикла называется ***итерацией***. А переменная ***i*** в данном примере называется ***счетчиком***.

Тот же код можно реализовать с использованием цикла ***for***:

```
for i in range (0,3,1):
    dType.SetPTPCmd(api, 2, 260, 60, 0, 0, isQueued=0)
    dType.SetPTPCmd(api, 2, 260, -60, 0, 0, isQueued=0)

```

*Рисунок III-3. Цикл **for** в Python*

Обратите внимание на то, что в этом примере не пришлось специальным образом объявлять переменную-счетчик ***i***. В данном примере для каждой итерации ***i*** примет значения 0, 1, 2, но не примет значение 3. Таким образом, цикл будет выполнен три раза. Это объясняется действием функции ***range***, которая генерирует последовательность чисел по правилу 3S:

range (Start, Stop, Step).

Start — это первое число в последовательности, в нашем случае это 0. ***Step*** — это шаг последовательности или разность между предыдущим и последующим членом. Каждое следующее число — это предыдущее число плюс ***Step***. Немного сложнее обстоят дела с ***Stop*** — это число, которое ограничивает последовательность. Но если первое число ***Start*** входит в сгенерированную последовательность, то число ***Stop*** в ней содержаться не будет. Именно поэтому ***i*** в нашем примере и не примет значение 3.

Воспользуемся циклами для решения задач, связанных с настройкой лазерного гравера Dobot.

Первая задача связана с определением необходимого времени для выжигания одной точки. Нам предстоит создать программу, в которой Dobot будет перемещать рабочий инструмент от точки к точке, осуществляя выжигание. При этом интервалы включения и выключения лазера в каждой точке будут увеличиваться.

```
i = 0
while i < 200:
    current_pose = dType.GetPose(api)
    dType.SetPTPCmdEx(api, 2, (120 + i),  0,  0, current_pose[3], 1)
    dType.SetEndEffectorLaserEx(api, 1, 100, 1)
    print(i)
    dType.dSleep(i)
    dType.SetEndEffectorLaserEx(api, 0, 100, 1)
    i = i + 1
```

Рисунок III-4. Выжигание лазером отверстий с интервалом в 1 мм

В самом начале программы объявляется переменная ***i*** — она является целым числом и равна 0. Цикл ***while*** во второй строке выполняется до тех пор, пока эта переменная меньше 200. При каждой итерации рабочий инструмент смещается на 1 мм и производит включение лазера, а затем через

паузу в ***i*** миллисекунд осуществляется выключение лазера. В 9 строке ***i*** увеличивается на 1. В результате будет выжжено 200 точек различной «яркости», как показано на рисунке III-5. Подобную процедуру имеет смысл проводить каждый раз для того, чтобы определиться, какой интервал будет самым подходящим для обрабатываемой поверхности.

Рисунок III-5. Выжигание точек с различным интервалом работы лазера

Вторая задача связана с выжиганием прямых линий. Толщина и глубина выжигаемой линии так же, как и продолжительность выжигания, зависит от скорости перемещения рабочего инструмента. Нам необходимо подобрать оптимальную скорость так, чтобы, с одной стороны, эта работа не занимала слишком много времени, а с другой — чтобы выжигание происходило на необходимую глубину. Воспользуемся циклом ***for***, как показано на рисунке ниже.

Рисунок III-6. Программа для выжигания линий при различных скоростях и ускорениях рабочего инструмента Dobot

В цикле будет осуществлено 20 повторений. Каждый раз к начальной точке повторения Dobot возвращается со скоростью и ускорением 20, как следует из строки 2. После включения лазера рабочий инструмент движется в конечную точку. Получаются наклонные линии. Это объясняется тем, что начальные точки имеют координаты ***120 + I · 10***, как это видно из строки 3, а конечные — ***120 + i***, как это видно из строки 6.

В результате получена следующая картина:

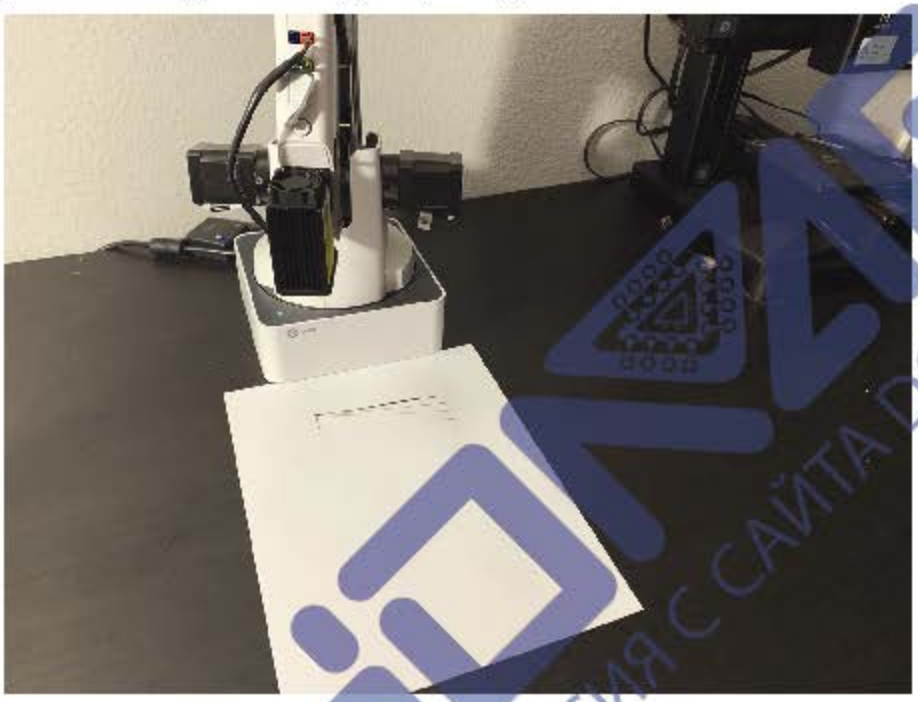


Рисунок III-7. Выжигание Dobot серии наклонных линий с различными скоростями и ускорениями рабочего инструмента

Обратите внимание на то, что при скорости и ускорении, равных 1 и 2, бумага прорезана насквозь, при скорости 3 — остается ровная черная линия, которая наиболее удобна при переносе чертежей на бумагу или картон. Наклонные линии выбраны не случайно. Dobot значительно проще перемещается в том случае, если работает только один из его моторов.

Мы провели два теста, однако, для того чтобы полностью использовать и понимать потенциал Dobot, вам необходимо провести еще серию тестов.

Задание для самостоятельной работы:

1) проверить, как зависит глубина прожигания от мощности лазера. Помните, что за мощность лазера отвечает третий аргумент в функции ***dType.SetEndEffectorLaserEx(api, 0, 100, 1)***;
2) определить, как влияет на печатание линий изменение двух параметров — скорости и ускорения рабочего инструмента, а также возможное изменение одного из этих параметров.

ЗАДАНИЕ IV. Вложенные циклы в Python

В дальнейшем нами будет построено множество фигур и алгебраических функций при помощи Dobot. Для того чтобы эти фигуры легче «читались», необходимо расчерчивать координатную плоскость, или сетку координат. Каким образом это сделать? Можно разлиновать лист по горизонтали и вертикали так, чтобы получилась серия прямоугольников или квадратов. Или разбить лист на квадраты, обозначив точками только их вершины. Попробуем для начала способ с точками. Начиная с этого раздела, мы будем программировать только в среде ***IDLE Python*** (о том, как ее настроить, мы рассказывали в разделе I). Первым шагом будет реализация одного столбца из точек, как показано на рисунке IV-1. Наша программа находится в строках с 25 по 34. До 25 строки происходит проверка, подключен ли Dobot к компьютеру, а также осуществляется операция ***Home***.

В 25 строке предусмотрен вывод сообщения ***START***, для того чтобы проще было контролировать выполнение всех операций. С этой же целью в 34 строке предусмотрен вывод ***FINISH***.

В 27 строке используется цикл ***for***. Переменная ***x*** последовательно примет все значения от 122 до 322 включительно с шагом в 1. В каждой итерации рабочий инструмент будет перемещаться к точке с координатой ***x***, –100, 0, включать лазер, ожидать одну десятую секунды, выключать лазер.

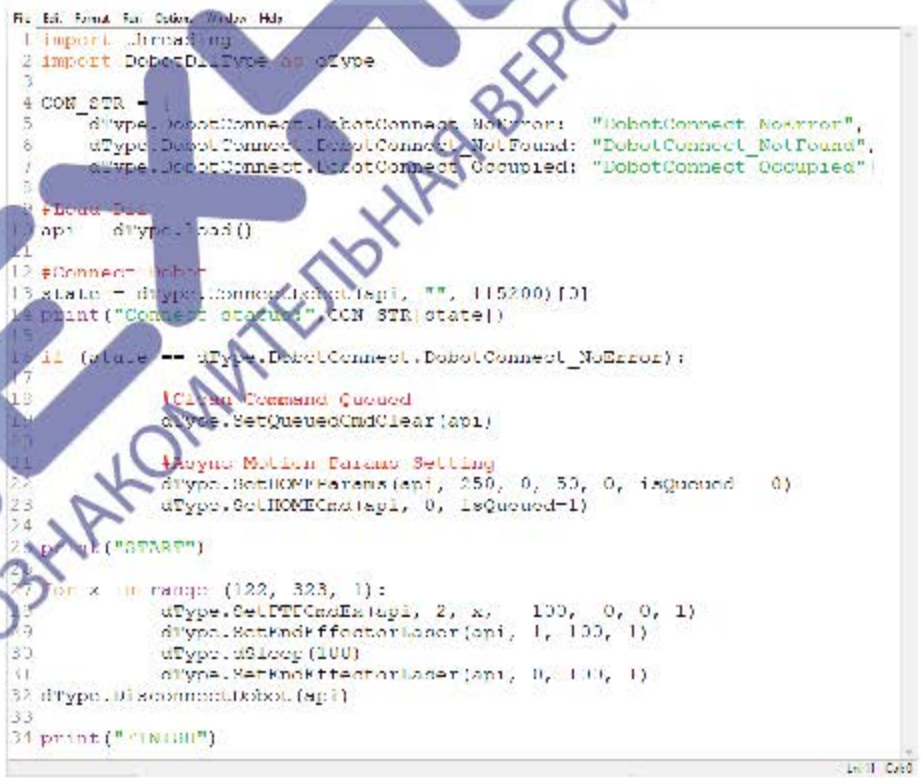


Рисунок IV-1. Программа для выжигания вертикального ряда точек с интервалом в 1 мм

В результате выполнения программы получится следующая картина:

Рисунок IV-2. Выжигание ряда точек с интервалом в 1 мм

Такая привязка не слишком удобна из-за того, что точки расположены очень близко друг к другу. Лучше, если интервал между точками будет составлять 10 мм. Для этого необходимо внести изменения в строку 27 и сделать параметр ***Step*** функции ***range*** в Python равной 10.

```
1 import threading
2 import Dobot.DllType as dType
3
4 CON_STR = {
5     dType.DobotConnect.DobotConnect_NoError:  "DobotConnect_NoError",
6     dType.DobotConnect.DobotConnect_NotFound: "DobotConnect_NotFound",
7     dType.DobotConnect.DobotConnect_Occupied: "DobotConnect_Occupied"}
8
9 #Load Dll
10 api = dType.load()
11
12 #Connect Dobot
13 state = dType.ConnectDobot(api, "", 115200)[0]
14 print("Connect status:",CON_STR[state])
15
16 if (state == dType.DobotConnect.DobotConnect_NoError):
17
18         #Clean Command Queued
19         dType.SetQueuedCmdClear(api)
20
21         #Async Motion Params Setting
22         dType.SetHOMEParams(api, 250, 0, 50, 0, isQueued = 0)
23         dType.SetHOMECmd(api, 0, isQueued=1)
24
25 print("START")
26
27 for x in range (122, 323, 10):
28         dType.SetPTPCmdEx(api, 2, x,  -100,  0, 0, 1)
29         dType.SetEndEffectorLaser(api, 1, 100, 1)
30         dType.dSleep(100)
31         dType.SetEndEffectorLaser(api, 0, 100, 1)
32 dType.DisconnectDobot(api)
33
34 print("FINISH")
```

Рисунок IV-3. Программа для выжигания лазером вертикального ряда точек с интервалом в 10 мм

Для того чтобы напечатать сетку точек, необходимо в каждом ряду печатать не одну точку, а целый ряд, при этом каждый вертикальный ряд точек печатать несколько раз, как это показано на рисунке IV-4. В этой программе используются два цикла один в другом. В первом изменяется переменная ***x***, во втором — ***y***. На каждое из 21 значения ***x*** приходится 21 значение ***y***. В 29 строчке происходит перемещение рабочего инструмента к точке с координатами ***x, y, 0***. В 30, 31, 32 — осуществляется выжигание этой точки.

```
File Edit Format Run Options Window Help
import threading
import DobotDllType as dType

CON_STR = {
    dType.DobotConnect.DobotConnect_NoError:  "DobotConnect_NoError",
    dType.DobotConnect.DobotConnect_NotFound: "DobotConnect_NotFound",
    dType.DobotConnect.DobotConnect_Occupied: "DobotConnect_Occupied"}

#Load Dll
api = dType.load()

#Connect Dobot
state = dType.ConnectDobot(api, "", 115200)[0]
print("Connect status:",CON_STR[state])

if (state == dType.DobotConnect.DobotConnect_NoError):

        #Clean Command Queued
        dType.SetQueuedCmdClear(api)

        #Async Motion Params Setting
        dType.SetHOMEParams(api, 250, 0, 50, 0, isQueued = 0)
        dType.SetHOMECmd(api, 0, isQueued=1)

print ("START")

for x in range (122, 323, 10):
        for y in range (-100, 110,10):
                dType.SetPTPCmdEx(api, 2, x,  y,  0, 0, 1)
                dType.SetEndEffectorLaser(api, 1, 100, 1)
                dType.dSleep(100)
                dType.SetEndEffectorLaser(api, 0, 100, 1)
dType.DisconnectDobot(api)

print ("FINISH")
```

Рисунок IV-4. Программа для выжигания сетки точек с интервалом в 10 мм

В результате выполнения программы будет получена сетка точек с шагом в 10 мм.

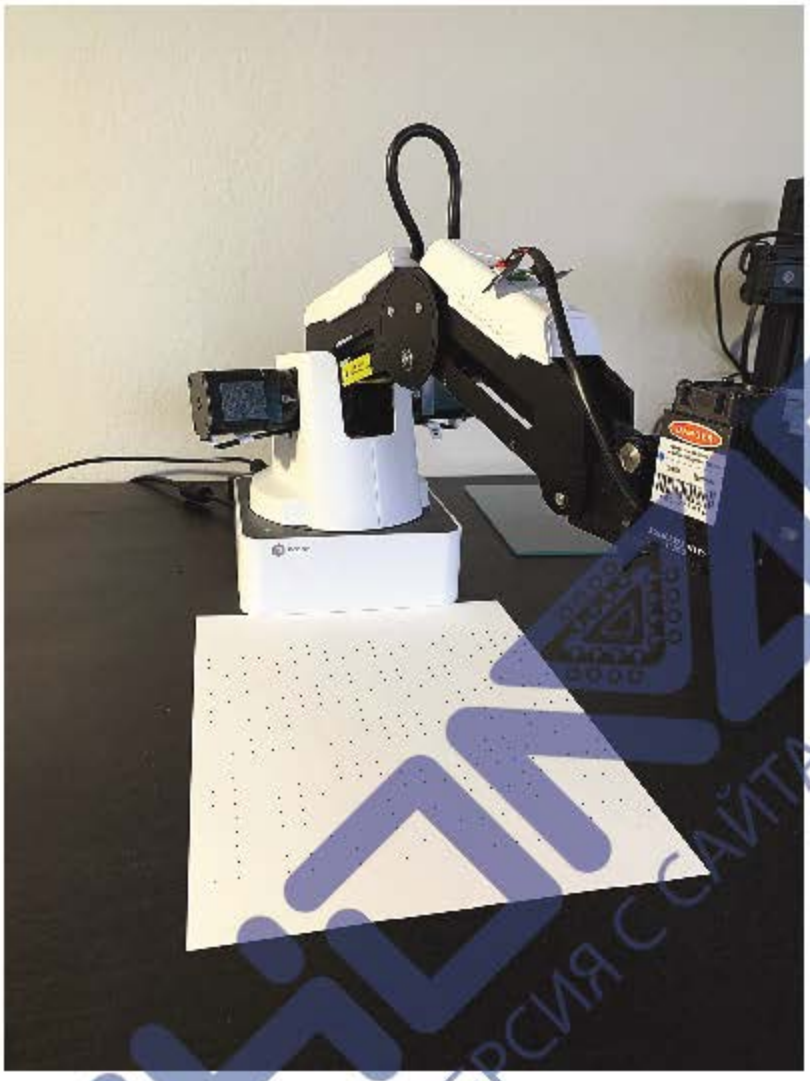
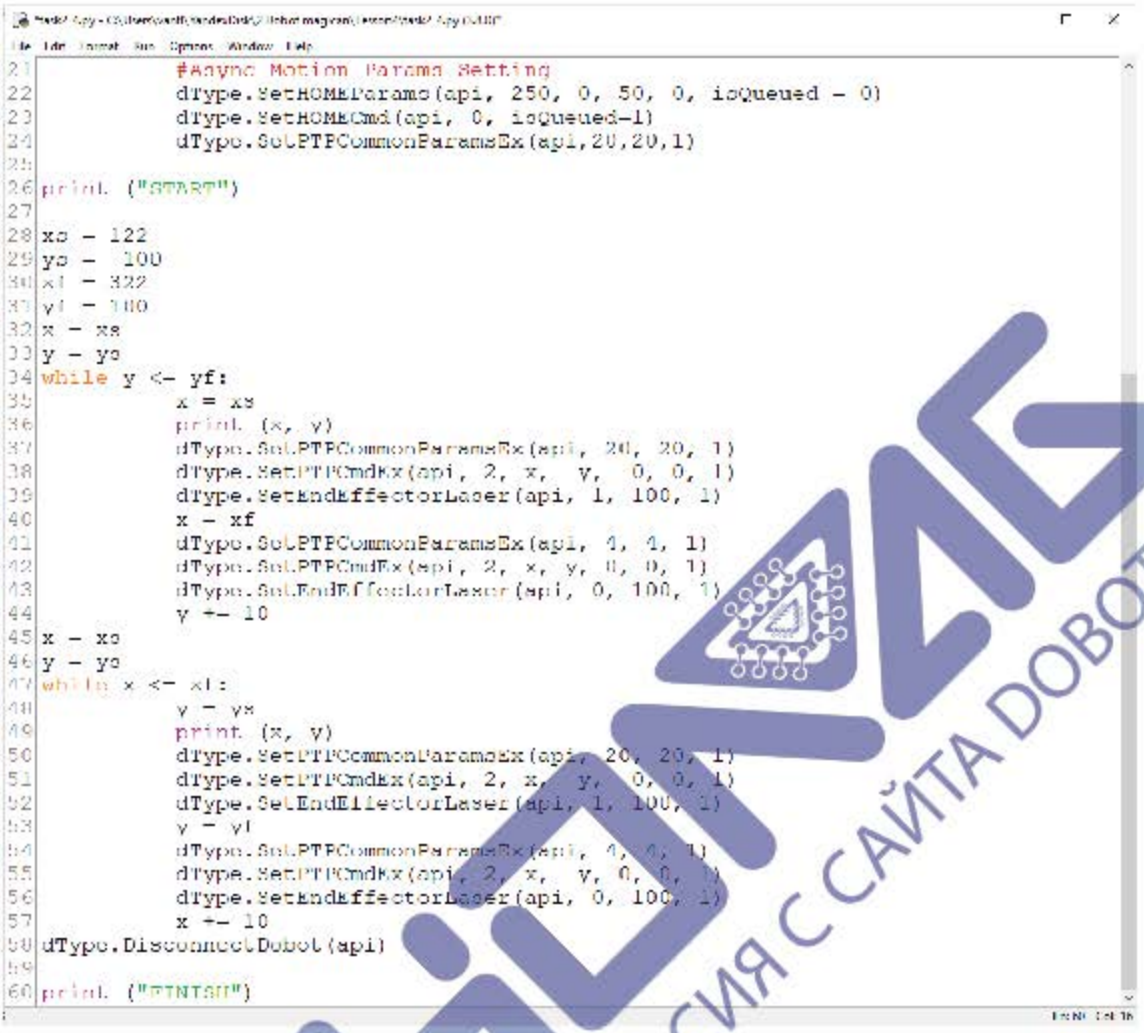


Рисунок IV-5. Сетка точек с шагом в 10 мм

Для того чтобы начертить клетки, понадобится разлиновать лист как по горизонтали, так и по вертикали. Программа для реализации этой задачи представлена на рисунке IV-6. Строки с 28 по 31 описывают ограничение области печати. В 32 и 33 строке переменным ***x*** и ***y*** присваиваются значения ***122*** и ***–100***. Это положение самой ближней и самой правой относительно Dobot линии.

Первый цикл в строке 34 необходим для выжигания вертикальных линий. Для этого в каждой итерации переменной ***x*** присваивается значение ***xs***, затем для контроля за выполнением программы выводятся на экран значения, характеризующие положение рабочего инструмента ***x*** и ***y***.



```
            #Async Motion Params Setting
            dType.SetHOMEParams(api, 250, 0, 50, 0, isQueued = 0)
            dType.SetHOMECmd(api, 0, isQueued=1)
            dType.SetPTPCommonParamsEx(api,20,20,1)

print ("START")

xs = 122
ys =  100
xf = 322
yf = 100
x = xs
y = ys
while y <= yf:
            x = xs
            print (x, y)
            dType.SetPTPCommonParamsEx(api, 20, 20, 1)
            dType.SetPTPCmdEx(api, 2, x,  y,  0, 0, 1)
            dType.SetEndEffectorLaser(api, 1, 100, 1)
            x = xf
            dType.SetPTPCommonParamsEx(api, 4, 4, 1)
            dType.SetPTPCmdEx(api, 2, x, y, 0, 0, 1)
            dType.SetEndEffectorLaser(api, 0, 100, 1)
            y += 10
x = xs
y = ys
while x <= xf:
            y = ys
            print (x, y)
            dType.SetPTPCommonParamsEx(api, 20, 20, 1)
            dType.SetPTPCmdEx(api, 2, x,  y,  0, 0, 1)
            dType.SetEndEffectorLaser(api, 1, 100, 1)
            y = yf
            dType.SetPTPCommonParamsEx(api, 4, 4, 1)
            dType.SetPTPCmdEx(api, 2, x,  y, 0, 0, 1)
            dType.SetEndEffectorLaser(api, 0, 100, 1)
            x += 10
dType.DisconnectDobot(api)

print ("PRINTED")
```

Рисунок IV-6. Программа для разлиновки горизонтальными и вертикальными линиями с шагом в 10 мм

В 50 строчке задается скорость перемещения Dobot к первой точке выжигания линии. В 51 строке происходит перемещение рабочего инструмента. В 52 строке включается лазер. В 53 строке переменной ***y***, которая отвечает за соответствующую координату рабочего инструмента, присваивается значение ***yf***. В строке 54 скорость рабочего инструмента замедляется, а в 55 строке он со включенным лазером перемещается. В 56 строке лазер выключается, в 57 строке, для того чтобы в следующей итерации производить выжигание новой линии, координата ***y*** увеличивается на 10 мм. Цикл будет повторяться, пока ***y*** не достигнет значения ***yf***, которое равно 100, — таким образом будет выжжена 21 линия.

Подобным образом будут выжжены и горизонтальные линии в цикле в строках с 47 по 57.

Задание для самостоятельной работы: написать код, который разлинует лист по диагонали.

ЗАДАНИЕ V. Ветвления

Важной возможностью любого языка программирования является использование ветвлений. До этого нами были разобраны программы, которые либо выполнялись линейно, то есть одна команда за другой, либо в цикле, но внутри цикла команды все так же выполнялись линейно. Ветвления же необходимы в том случае, если последовательность действий зависит от того, какие значения принимает какая-либо переменная. В языке Python ветвления реализуются при помощи структуры ***if — else — else-if***. Рассмотрим пример кода, в котором необходимо вывести числа от 0 до 10 и указать, четное или нечетное каждое из них.

Рисунок V-1. Код программы, выводящей четное или нечетное число ***i***

Как видно из строки 1, переменная ***i*** примет значения от 0 до 10 с шагом в 1. Для каждого из этих значений осуществляется проверка — логическая операция в строке 2. По сути, после ***if*** задается вопрос, на который возможны только два ответа: ***Истина (True)*** или ***Ложь (False)***. В данном случае выражение ***i%2 == 0*** означает: остаток от деления на 2 равен нулю? Если да, то выполняется код, который находится в строках после двоеточия во второй строке. Обратите внимание, что этот код должен находиться на расстоянии в 4 пробела или одной табуляции от линии, проходящей вниз от буквы ***i*** оператора ***if***. Если же условие ***i%2 == 0*** не выполнено, то выполняется часть кода, находящаяся под ***else***. В этом случае также необходимо соблюдать расстояние в 4 пробела или одну табуляцию.

Важно отметить, что за одну итерацию может быть выполнена либо 3, либо 5 строка. Именно поэтому эту конструкцию и называют ветвлением. Любое многообразие вариантов можно представить в виде серии вложенных друг в друга ветвлений. Например, для того чтобы отгадать число от 1 до 32, достаточно задать 5 вопросов с парным выбором. Это так называемый метод половинного выбора. Первый вопрос будет «Это число больше 16?», второй — «Это число больше 8?» и т.д.

Кроме того, в языке Python, как и в большинстве других языков программирования, существует возможность реализовать множественные ветвления:

```
File  Edit  Format  Run  Options  Window  Help
for i in range (0,11,1):
        if i%2 == 0:
                print(str(i) + " - Четное")
        elif i     5:
                print(str(i) + "   Так это же 5!")
        else:
                print(str(i) + " - Нечетное")

                                                    Ln: 4  Col: 21
```

Рисунок V-2. Множественное ветвление в Python

В строке 4 осуществляется альтернативный выбор. Однако обратите внимание: он будет осуществлен только в том случае, если ложным окажется выражение во второй строке.

```
0 - Четное
1   Нечетное
2   Четное
3 - Нечетное
4 - Четное
5 - Так это же 5!
6   Четное
7   Нечетное
8 - Четное
9 - Нечетное
10 - Четное
```

Рисунок V-3. Результат выполнения кода на Рисунок V-2. Множественное ветвление в Python

Если бы в 4 строке было условие *i* == *4*, то надпись «Так это же 4!» не появилась бы на экране.

Настало время разработать ветвления для выжигания орнаментов Dobot. Первый орнамент — это «Змейка», ломаная линия:

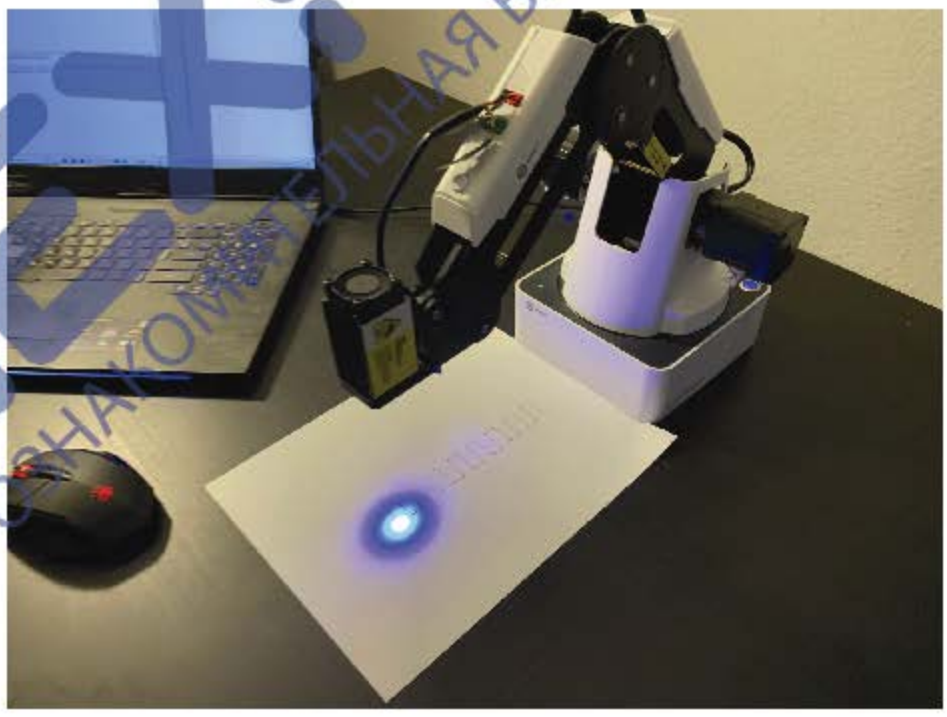


Рисунок V-4. Орнамент «Змейка», выжигаемый Dobot

Код для реализации этого орнамента представлен на рисунке V-5. Текущие координаты положения рабочего инструмента в данном коде ***x*** и ***y***. В 32 и 33 строках задается точка, на которой будет начато выжигание. В 35 строке Dobot двигается к этой координате. В 36 строке осуществляется уменьшение скорости и ускорения движения рабочего инструмента. В 37 строке включается лазер. В цикле ***for*** (строка 38) будет осуществлено 20 движений. В зависимости от того, является ли число ***i*** четным или нечетным, рабочий инструмент переместится или к точке с координатой ***y*** = ***–10***, или к точке с координатой ***y*** = ***10***.

```
26 print ("START")
27
28 xs   122
29 yn -  100
30 xf = 322
31 yf - 100
32 x   xs
33 y = 10
34 dType.SetPTPCommonParamsEx(api, 20, 20, 1)
35 dType.SetPTPCmdEx(api, 2, x,  y,  0, 0, 1)
36 dType.SetPTPCommonParamsEx(api, 4, 4, 1)
37 dType.SetEndEffectorLaser(api, 1, 100, 1)
38 for i in range (0,21,1):
39             print (x, y)
40             if i%2 == 1:
41                         y=10
42             else:
43                         y -10
44             x = x + 10
45             dType.SetPTPCmdEx(api, 2, x, y, 0, 0, 1)
46 dType.SetEndEffectorLaser(api, 0, 100, 1)
47
48 print ("FINISH")
```

Рисунок V-5. Код для выжигания орнамента «Змейка» при помощи Dobot

Следующий орнамент «Зигзаг». Он получается после несложной модернизации предыдущего кода.

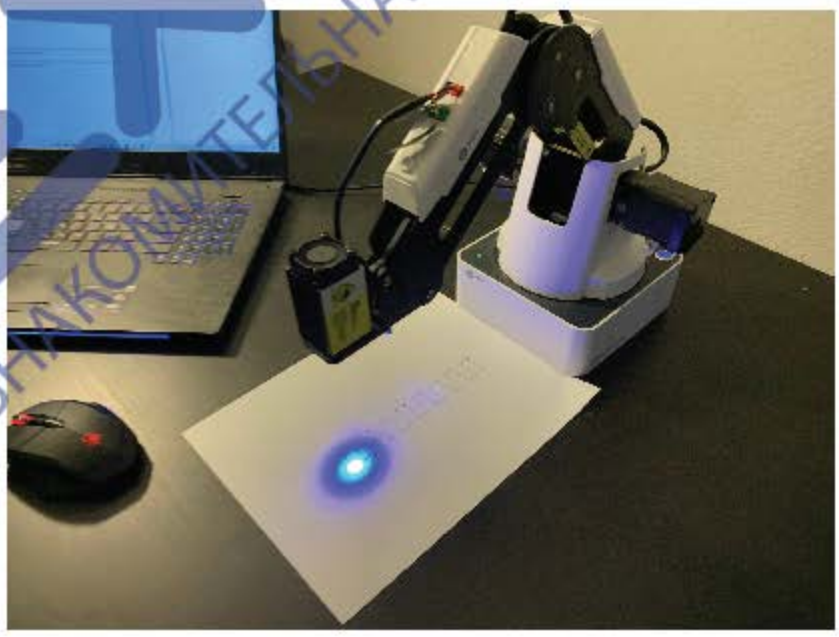


Рисунок V-6. Орнамент «Зигзаг»

В строке 44 добавлено перемещение по оси ***x*** до того, как реализовано движение по оси ***y***.

```
28 xs = 122
29 ys = -100
30 xf = 322
31 yf = 100
32 x = xs
33 y = 10
34 dType.SetPTPCommonParamsEx(api, 20, 20, 1)
35 dType.SetPTPCmdEx(api, 2, x, y, 0, 0, 1)
36 dType.SetPTPCommonParamsEx(api, 4, 4, 1)
37 dType.SetEndEffectorLaser(api, 1, 100, 1)
38 for i in range (0,22,1):
39             print (x, y)
40             if i%2 == 1:
41                     y=10
42             else:
43                     y= 10
44             dType.SetPTPCmdEx(api, 2, x, y, 0, 0, 1)
45             x = x + 10
46             dType.SetPTPCmdEx(api, 2, x, y, 0, 0, 1)
47 dType.SetEndEffectorLaser(api, 0, 100, 1)
48
49 print ("FINISH")
```

Рисунок V-7. Код для орнамента «Зигзаг»

На рисунке V-8 представлены несколько замкнутых линий, которые, в свою очередь, состоят из прямых. Каждая прямая перпендикулярна соседней. Все фигуры состоят из 4 одинаковых частей (обратите внимание, что в каждой из частей обязательно нечетное количество линий). Давайте условно разберем весь орнамент на четверти. В первой четверти на любом шаге увеличивается и координата ***x*** и координата ***y***. Во второй четверти увеличивается ***y***, в то время как ***x*** уменьшается. В третьей четверти уменьшаются обе координаты. В четвертой уменьшается только ***y***. Назовем одно изменение координаты шагом (в нашем примере будет два шага: ***xd***, ***yd*** — для соответствующих осей). Их значение будет неизменно в течение всей программы, а вот знак при переходе от одной четверти к другой будет меняться. Кроме того, для того чтобы иметь возможность выжигать не одну, а несколько ломаных линий, необходимо предусмотреть возможность изменения количества линий в одной четверти. На рисунке V-8 видно 4 орнамента с общим количеством прямых линий: 12, 28, 36, 52.

Рисунок V-8. Серия замкнутых прямоугольных орнаментов

Рассмотрим код для выжигания прямоугольных орнаментов (рисунок V-9). В 27 строке переменной ***K*** присваивается значение, равное количеству прямых линий в орнаменте, — это обязательно нечетное число, умноженное на 4. В 32 строке определяется координата ***x*** самой нижней части рисунка, она зависит от количества прямых линий, из которых он состоит, а также от длины этих линий. Уменьшаемое в этой формуле — это центр нашего рабочего поля, вычитаемое расстояние от центра орнамента до края. В 34 и 35 строках задаются шаги — длины прямых, параллельных горизонталям и вертикалям. В цикле происходит смещение рабочего инструмента или по горизонтали, или по вертикали ***K*** раз (44–47 строки). По достижении второй четверти шаг ***yd*** меняет знак (строки 48–49). Для третьей четверти знак меняется у ***xd*** (строки 50–51). В строках 52–53 знак ***yd*** изменяется на первоначальный.

```
print ("START")
K = 52
xs = 122
ys = -100
xf = 322
yf = 100
x = (xs + xf) / 2    (K*10/4)/2
y = 0
xd = 10
yd = 10

dType.SetPTPCommonParamsEx(api, 20, 20, 1)
dType.SetPTPCmdEx(api, 2, x,  y,  0, 0, 1)
dType.SetPTPCommonParamsEx(api, 1, 1, 1)


for i in range (1, K+1, 1):
            print (x, y)
            if i % 2 == 0:
                        x = x + xd
            else:
                        y = y + yd
            if i == K // 4:
                        yd = -1 * yd
            if i == K // 2:
                        xd = -1 * xd
            if i == K // 2 + K // 4:
                        yd = 1 * yd

            dType.SetEndEffectorLaser(api, 1, 100, 1)
            dType.SetPTPCmdEx(api, 2, x, y, 0, 0, 1)

dType.SetEndEffectorLaser(api, 0, 100, 1)
print ("FINISH")
```

Рисунок V-9. Код программы для выжигания прямоугольных орнаментов

В строке 55 происходит включение лазера. В 56 строке перемещение рабочего инструмента в позицию, вычисленную ранее.

Задание для самостоятельной работы: реализовать орнаменты, связанные с ветвлениями, например «Косичка».

ЗАДАНИЕ VI. Списки в Python. Случайные блуждания

Одним из самых больших достоинств языка программирования Python является его работа с данными. Мы уже упоминали выше о реализации списков в этом языке программирования. Список — это набор данных, к которым можно обращаться по индексу. То есть каждому элементу списка присваивается его номер. Тип данных, который хранится в списке, может быть любым.

```
>>> a = [1,2,3,4,5]
>>> print (a)
[1, 2, 3, 4, 5]
>>>
```

*Рисунок VI-1. Создание списка **a** и вывод его на экран*

На рисунке VI-1 создается список из 5 элементов. Для этого они все перечисляются внутри квадратных скобок через запятую. Для того чтобы добавить элемент в список, необходимо воспользоваться методом ***append***, как это показано на рисунке VI-2. Обратите внимание на синтаксис методов. Сначала они записываются после точки, которая, в свою очередь, записывается после обьекта, к которому применяется метод.

```
>>> a = [1,2,3,4,5]
>>> print (a)
[1, 2, 3, 4, 5]
>>> a.append(6)
>>> print(a)
[1, 2, 3, 4, 5, 6]
>>>
```

*Рисунок VI-2. Метод **append** в языке программирования Python*

В данном случае к списку ***a*** применяется метод ***append***, который добавляет к нему число 6.

```
>>> a = [1,2,3,4,5]
>>> print (a)
[1, 2, 3, 4, 5]
>>> a.append(6)
>>> print(a)
[1, 2, 3, 4, 5, 6]
>>> print (a[2])
3
>>>
```

*Рисунок VI-3. Вызов второго элемента списка **a***

На рисунке VI-3 показан вызов второго элемента списка. Обратите внимание на то, что это число 3, а не 2, как вы это, возможно, ожидали. Это связано с тем, что элементы списка нумеруются, начиная с нулевого элемента.

У списков в Python есть множество методов, подробнее о них мы будем говорить каждый раз, как они понадобятся нам для реализации идей.

Настало время создать новую задачу для Dobot. Нам необходимо осуществить так называемые случайные блуждания. Рабочий инструмент должен будет перемещаться в случайном направлении, выжигая при каждом движении прямую линию. Для реализации этого замысла нам понадобится подключить библиотеку ***random***:

```
1 import threading
2 import DobotDllType as dType
3 import random as rnd
```

*Рисунок VI-4. Подключение библиотеки **random** в Python*

Из библиотеки ***random*** нам понадобится функция ***randint (Start, Stop)***: она возвращает любое случайное значение в интервале от ***Start*** до ***Stop*** включительно.

В строках кода 32–45, показанного на рисунке VI-5, реализуется заполнение списков ***x*** и ***y*** для случайных блужданий. В строке 35 осуществляется проверка того, чтобы блуждание на каждом из шагов не было равно 0 и по оси ***x***, и по оси ***y*** одновременно. В 36 и 37 строках происходит случайный выбор направления блуждания.

```
26 print ("START")
27
28 x = [222]
29 y = [0]
30 x1=0
31 y1 0
32 while (True):
33             x1=0
34             y1=0
35             while (x1 == 0 and y1 == 0):
36                     x1 = rnd.randint(0,10)
37                     y1 = rnd.randint(0,10)
38             xa = rnd.randint(-1,1)
39             ya = rnd.randint(-1,1)
40             x.append(x[-1] + xa*x1)
41             y.append(y[-1] + ya*y1)
42             if (x[-1] > 222 or y[-1]>100 or x[-1] < 122 or y[-1] < -100):
43                     del (x[-1])
44                     del (y[-1])
45                     break
46 print (len(x))
47 dType.SetPTPCommonParamsEx(api, 20, 20, 1)
48 dType.SetPTPCmdEx(api, 2, x[0], y[0],  0, 0, 1)
49 dType.SetPTPCommonParamsEx(api, 4, 4, 1)
50 dType.SetEndEffectorLaser(api, 1, 100, 1)
51 for i in range (1, len (x)):
52     print (i, x[i], y[i])
53     dType.SetPTPCmdEx(api, 2, x[i], y[i],  0, 0, 1)
54 dType.SetEndEffectorLaser(api, 0, 100, 1)
55 dType.DisconnectDobot(api)
56
57 print ("FINISH")
```

Рисунок VI-5. Код для реализации случайных блужданий

Для того чтобы понять, как это происходит, обратимся к примеру (рисунок VI-6). В данном случае рабочий инструмент находится в точке с координатами $x = 240$ и $y = 30$. Затем переменные ***xl*** и ***yl*** принимают значения **4** и **4**. А переменные ***xa*** и ***ya*** значения **1** и **–1**. Следовательно, перемещение рабочего инструмента произойдет таким образом, что его координата по ***x*** увеличится на **4**, а координата по ***y*** уменьшится на **4**. Именно значения текущих координат и будут записаны в списки ***x*** и ***y*** в строках 40 и 41. На следующем шаге координата ***y*** осталась неизменной, а координата ***x*** изменилась на **6** в отрицательную сторону.

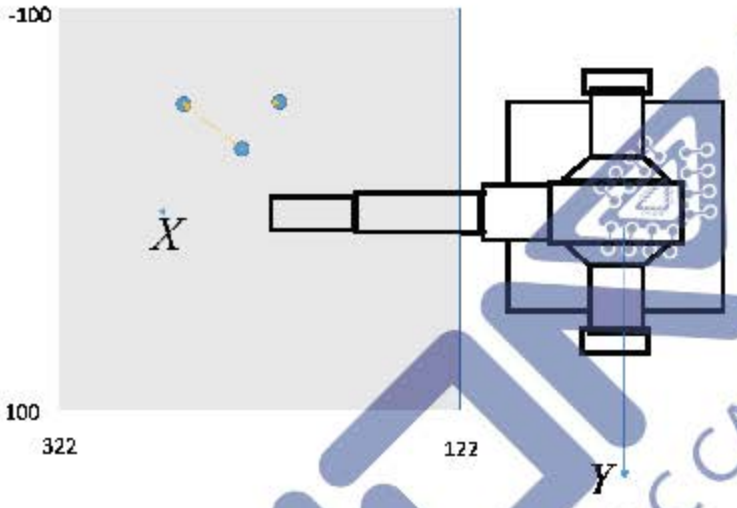


Рисунок VI-6. Два случайных блуждания

Подобные блуждания будут продолжаться до тех пор, пока одна из координат инструмента не окажется за пределами рабочей области. В этом случае после ветвления в строке 42 произойдет стирание последних записей из списков (строки 43, 44) и выход из цикла (строка 45).

В строке 48 осуществляется перемещение Dobot в центр рабочей области. В 49 — уменьшение скорости и ускорения передвижения рабочего инструмента. В 50 — включается лазер. В цикле в строках 51–53 осуществляется обход всех точек случайного блуждания. При каждом запуске будут получаться различные результаты — с разным количеством и направлением перемещений, но в итоге получится результат, подобный представленному на рисунке VI-7.

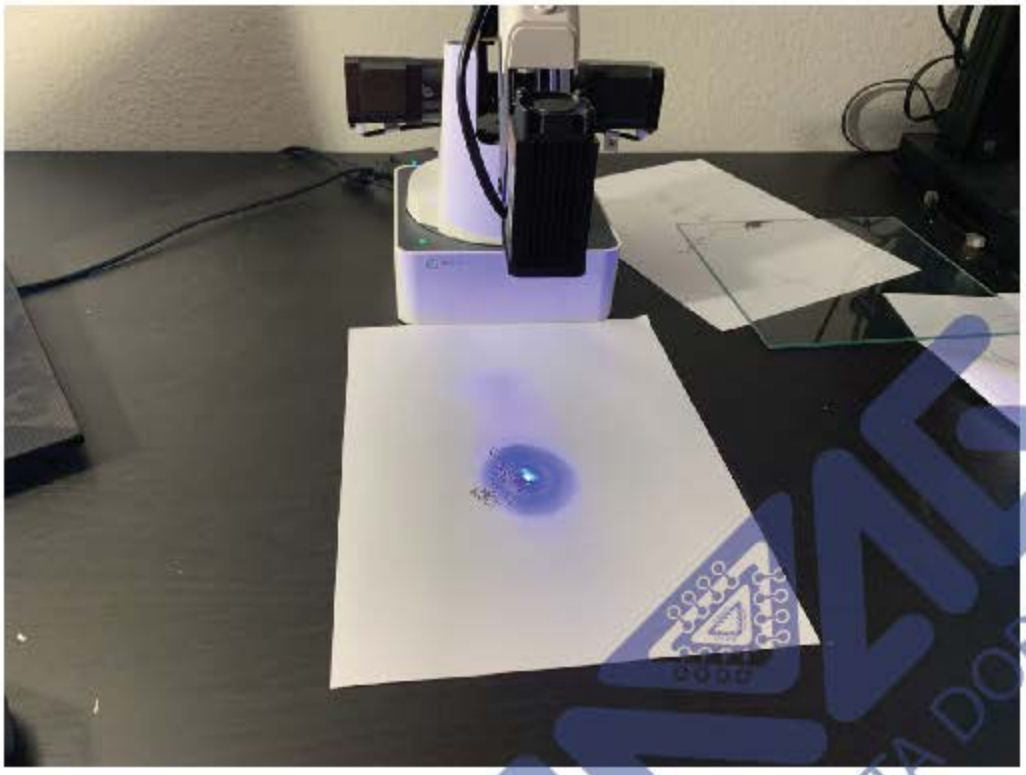


Рисунок VI-7. Случайные блуждания рабочего инструмента Dobot

Задание для самостоятельной работы: реализовать код, осуществляющий случайные блуждания, таким образом, чтобы за одно перемещение рабочего инструмента изменялась только одна координата.

ЗАДАНИЕ VII. Функции в Python

Одним из удобств языка Python является возможность использовать так называемые функции. Давайте обратимся к примеру:

```
1 def arithmetic (a,b):
2             """Осуществляет все арифметические операции с a и b"""
3             print ("sum - " + str(a + b))
4             print ("difference = " + str(a - b))
5             print ("multiplicate = " + str(a * b))
6             print ("division   " + str(a // b))
7             return "done"
8
9 print (arithmetic(10,2))
```

Рисунок VII-1. Пример использования функций в языке Python

В первой строке объявляется функция. Ее имя ***arithmetic***, она принимает два аргумента: ***a*** и ***b***. Тело функции располагается в строках 2–7. Действие этой функции — вывод на экран всех арифметических операций. В 7 строке располагается оператор ***return,*** который возвращает значение функции. В данном случае оно будет всегда одно и то же — ***done***. Обратите внимание на то, что все строки, которые относятся к телу функции, записаны с отступом в 4 пробела или 1 табуляцию. Кроме того, после объявления функции и ее аргументов в первой строке находится двоеточие.

При вызове функции в строке 9 осуществляется выполнение кода (вывод на экран всех операций функции), сам же оператор ***print*** из 9 строки выводит возвращаемое функцией значение — ***done***.

Такой подход к программированию приводит к повышению читаемости кода, значительно сокращает его длину, а следовательно, и количество ошибок.

Воспользуемся данным способом для того, чтобы сделать программирование Dobot более удобным.

Какие базовые функции нам могут понадобиться?

1. Выжигание точки в текущих координатах рабочего инструмента.
2. Перемещение рабочего инструмента из текущей точки в заданную.
3. Выжигание прямой при перемещении рабочего инструмента из текущей точки в заданную.

Функции два и три можно объединить, ведь перемещение инструмента из точки в точку может осуществляться как с включенным лазером, так и с выключенным. Таким образом, включение/выключение лазера должно являться аргументом этой обобщенной функции. То же касается и интенсивности воздействия лазера на обрабатываемую поверхность.

Что касается функции выжигания в текущей точке, то ее аргументами должны быть интенсивность работы лазера и время его работы.

```
import threading
import DobotDllType as dType

def laser_point (power, time):
            """Выжигает отверстие в текущем положении рабочего инструмента"""
            dType.SetEndEffectorLaser(api, 1, power, 1)
            dType.dSleep(time)
            dType.SetEndEffectorLaser(api, 0, power, 1)

CON_STR = {
    dType.DobotConnect.DobotConnect_NoError:  "DobotConnect_NoError",
    dType.DobotConnect.DobotConnect_NotFound: "DobotConnect_NotFound",
    dType.DobotConnect.DobotConnect_Occupied: "DobotConnect_Occupied"}

#Load Dll
api = dType.load()

#Connect Dobot
state = dType.ConnectDobot(api, "", 115200)[0]
print("Connect status:",CON_STR[state])

if (state == dType.DobotConnect.DobotConnect_NoError):

            #Clean Command Queued
            dType.SetQueuedCmdClear(api)

            #Async Motion Params Setting
            dType.SetHOMEParams(api, 250, 0, 0, 0, isQueued = 0)
            dType.SetHOMECmd(api, 0, isQueued=1)
            dType.SetPTPCommonParamsEx(api,20,20,1)
print ("START")

laser_point(100,20)
dType.DisconnectDobot(api)

print ("FINISH")
```

Рисунок VII-2. Функция выжигания точки ***laser_point***

В строке 4 объявляется функция ***laser_point***, два ее аргумента используются для задания интенсивности выжигания и времени между включением и выключением лазера. В строке 7 используется функция **dType.dSleep**. Ее аргумент — это время в миллисекундах.

Перейдем к следующей функции, которая объединит перемещение рабочего инструмента с включенным лазером (выжигание) и с выключенным лазером. Код для реализации этой функции и ее вызова показан на рисунке VIII-3.

```
10 def laser_line (power, x, y, V   20, a   20):
11             """Выжигает линию от текущей координаты до координаты x y"""
12             dType.SetPTPCommonParamsEx(api, V, a, 1)
13             if power == 0:
14                         s = 0
15             else:
16                         s = 1
17             dType.SetEndEffectorLaser(api, s, power, 1)
18             dType.SetPTPCmdEx(api, 2, x, y, 0, 0, 1)
19             dType.SetEndEffectorLaser(api, 0, 0, 1)
20
21 CON_STR = {
22     dType.DobotConnect.DobotConnect_NoError:  "DobotConnect_NoError",
23     dType.DobotConnect.DobotConnect_NotFound: "DobotConnect_NotFound",
24     dType.DobotConnect.DobotConnect_Occupied: "DobotConnect_Occupied"}
25
26 #Load Dll
27 api = dType.load()
28
29 #Connect Dobot
30 state   dType.ConnectDobot(api, "", 115200)[0]
31 print("Connect status:",CON_STR[state])
32
33 if (state == dType.DobotConnect.DobotConnect_NoError):
34
35             #Clean Command Queued
36             dType.SetQueuedCmdClear(api)
37
38             #Async Motion Params Setting
39             dType.SetHOMEParams(api, 250, 0, 0, 0, isQueued = 0)
40             dType.SetHOMECmd(api, 0, isQueued 1)
41             dType.SetPTPCommonParamsEx(api,20,20,1)
42 print ("START")
43
44 laser_line(100, 300,  100, 3 , 3)
45 laser_line(0, 300, 100)
46
47 print ("FINISH")
```

*Рисунок VII-3. Функция **laser_line** для перемещения рабочего инструмента с выжиганием и без выжигания*

У функции ***laser_line*** пять аргументов:

1) **power** — это интенсивность работы лазера. Если этот аргумент равен нулю, то лазер будет выключен. Для реализации этого используется ветвление в строках 13–17. В результате выполнения этих ветвлений переменная ***s*** примет значение либо **0**, либо **1**. Таким образом, в 17 строке будет осуществлено либо включение, либо «не включение» лазера;
2) ***x*** — координата ***x***, в которой должен оказаться рабочий инструмент;
3) ***y*** — координата ***y***, в которой должен оказаться рабочий инструмент;
4) ***V*** — скорость перемещения рабочего инструмента, по умолчанию она равна 20;
5) ***a*** — ускорение перемещения рабочего инструмента, по умолчанию оно равно 20.

В 40 строке рабочий инструмент попадает в положение ***home.*** Затем в 44 строке осуществляется перемещение включенного лазера в место с координатами 300, –100 со скоростью и ускорением, равными 3. Затем лазер перемещается в точку с координатами 300, 100 с выключенным лазером и скоростью и ускорением, равным по умолчанию 20. Результат выполнения этой программы показан на рисунке VII-4. Обратите внимание, что на листе бумаги выжжена прямая из положения ***home*** в положение 300, –100, а затем рабочий инструмент сместился к другому краю листа без выжигания.

Рисунок VII-4. Выжигание при помощи Dobot прямой, а затем перемещение без выжигания

Задание для самостоятельной работы: переписать уже написанные коды для Dobot с использованием двух написанных в этом параграфе функций.

ЗАДАНИЕ VIII. Управление Dobot с клавиатуры

В этом задании необходимо создать программу для управления Dobot с клавиатуры компьютера. Для ввода текста с клавиатуры в языке программирования Python реализована функция ***input()***. Ее работа продемонстрирована ниже:

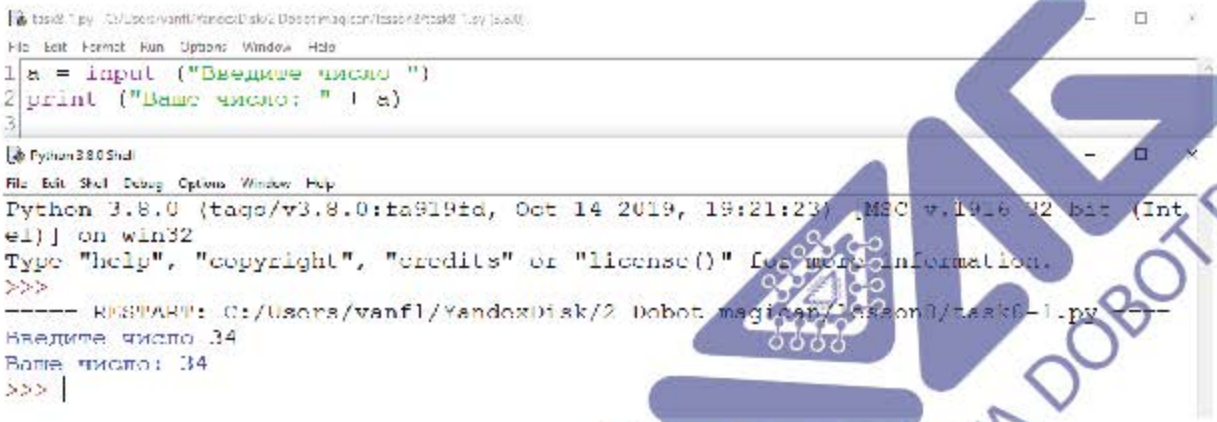


*Рисунок VIII-1. Реализация функции **input()** в языке Python*

В первой строке переменной ***a*** присваивается значение результата выполнения функции ***input()***. Аргументом этой функции является строка, которая будет выведена на экран перед вводом. При этом любое введенное значение автоматически имеет тип данных ***str***. Для того чтобы использовать другой тип данных, это значение необходимо преобразовать, если это возможно. Например, на рисунке VIII-2 в строке 2 происходит пребразование ***a*** в численный формат. Затем в третьей строке осуществляется умножение значения переменной ***a*** на 2. Результат выводится на экран. Обратите внимание на то, что сначала в языке программирования Python выполняются действия в скобках.

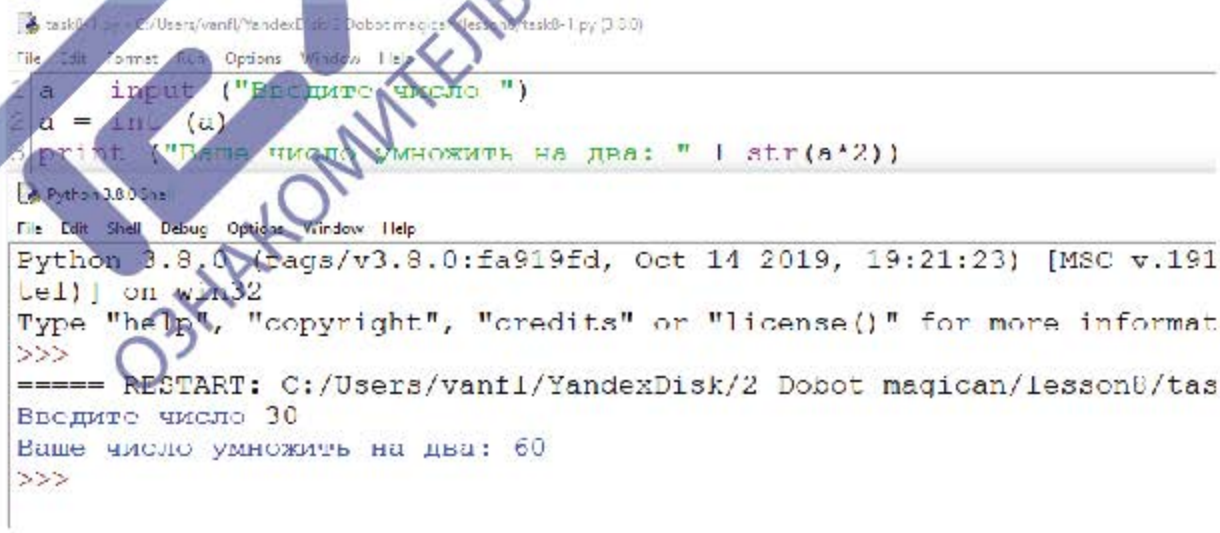


*Рисунок VIII-2. Преобразование строковой переменной **a** к формату целого числа*

Если же не осуществить преобразование переменной к численному формату, то в результате умножения на 2 получится следующий результат:

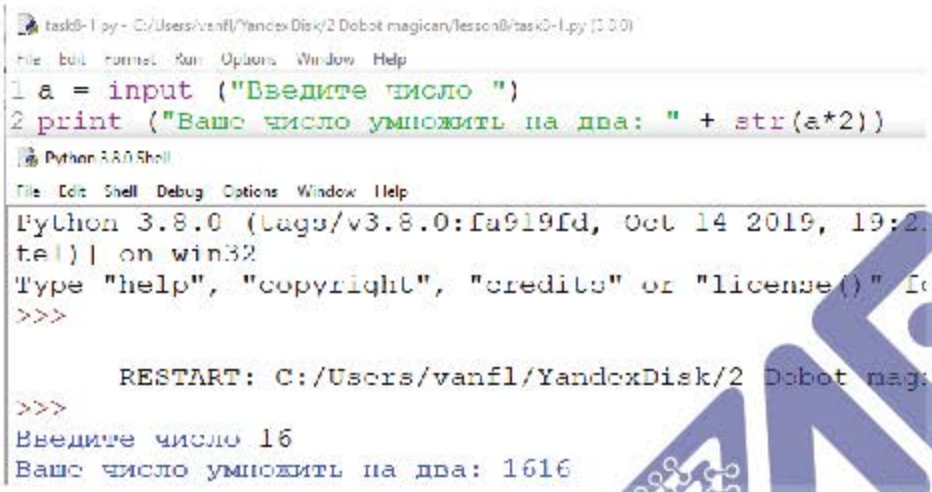


Рисунок VIII-3. Копирование строковых переменных в Python

В прошлом задании нами были разработаны две функции: для выжигания точки и перемещения рабочего инструмента. При этом перемещение инструмента могло происходить как с включенным лазером, так и с выключенным. В первом случае происходило выжигание прямой, во втором — нет.

В нынешнем задании необходимо создать пользовательский интерфейс. Во-первых, у пользователя необходимо спросить о том, что он собирается делать: выжигать отверстие или перемещаться в другую точку. Для выжигания в точке необходимо знать интенсивность и время выжигания. Для перемещения же необходимо узнать такие параметры, как новые координаты места перемещения, скорость и ускорение движения. В случае если перемещение будет осуществляться с включенным лазером, необходимо знать еще и интенсивность его работы.

Рассмотрим получившуюся программу по частям.

На рисунке VIII-4 показана функция ***main_menu()***. Ее задача — оповестить пользователя о возможных действиях, а также предоставить ему информацию о текущем положении рабочего инструмента при помощи функции ***report ()***.

*Рисунок VIII-4. Функция **main_menu()***

Действие функции ***report*** показано на рисунке VIII-5. Она выводит на экран текущие координаты рабочего инструмента. Для этого используется функция библиотеки Dobot

dType.GetPose(api),

которая возвращает значения координат рабочего инструмента и углов поворота шаговых двигателей Dobot. По сути, именно эти углы и задают положение рабочего инструмента.

```
14 def report ():
15         """Выводит информацию о текущих координатах"""
16         print ("Текущие координаты (x,y,z): (" + str (int(dType.GetPose(api)[0])) +
17                ", " + str (int(dType.GetPose(api)[1])) +
18                ", " + str(int((dType.GetPose(api)[2]))) + ")")
19
```

*Рисунок VIII-5. Функция **report()***

Функция ***report()*** будет вызываться при каждом запросе ввода от пользователя, для того чтобы он в любой момент знал о положении рабочего инструмента.

```
20 def one_whole ():
21         """Собирает данные о выжигании отверстия, возвращает время и интенсивность выжигания"""
22         while (1):
23                 print ("Выжигаем отверстие")
24                 report ()
25                 i = input ("Введите интенсивность выжигания от 0 до 100 ")
26                 try:
27                         i = abs(int (i))
28                         if i < 101:
29                                 break
30                 except:ValueError
31         while (1):
32                 print ("Выжигаем отверстие")
33                 report ()
34                 t = input ("Введите время выжигания в миллисекундах ")
35                 try:
36                         t = abs(int (t))
37                         break
38                 except:ValueError
39         return i,t
```

*Рисунок VIII-6. Функция для выжигания одного отверстия **one_whole()***

Функция ***one_whole()*** на рисунке VIII-6 спрашивает пользователя об интенсивности и времени выжигания.

Для того чтобы исключить ошибки пользователя при вводе, в коде применены следующие приемы.

С 22 по 30 строки в бесконечном цикле организуется опрос об интенсивности выжигания. Это число от 0 до 100. В случае если переменную ***i***, в которую записывается результат опроса, можно преобразовать к целочисленному типу, в строке 27 осуществляется ее проверка на то, что она меньше **101**. Если эта проверка пройдена, то в строке 29 осуществляется досрочное прерывание цикла.

В строках 26–30 используется конструкция ***try*** — ***except*** для обработки исключений. В блоке ***try*** мы выполняем инструкцию, которая может породить исключение, а в блоке ***except*** мы перехватываем его. В случае если пользователь ввел некорректное значение, например «фвавы», блок ***try-except*** перехватит его, при этом программа не будет остановлена и не будет выведено сообщение об ошибке. Кроме того, не будет осуществлен и выход из бесконечного цикла. Однако допускается ввод отрицательного значения интенсивности. Для этого к полученному цифровому значению применяется функция ***abs ()***, которая возвращает модуль аргумента.

В строках 31–38 осуществляется опрос, подобный описанному выше для времени выжигания. В строке 39 перечислены возвращаемые значения.

Функция ***move_line ()*** спрашивает пользователя о месте, к которому должен переместиться рабочий инструмент (рисунок VШ-7).

*Рисунок VIII-7. Функция **move_line ()** для перемещения рабочего инструмента*

В данной функции реализованы те же проверки для правильности ответов пользователя. Кроме того, осуществляются проверки на то, чтобы рабочий инструмент не выходил за пределы рабочей области.

Функция ***move_line ()*** в 79 строке возвращает значения координат, к которым будет перемещаться Dobot, а также значения скорости и ускорения этого перемещения.

Строки кода с 84 по 121 получены нами на предыдущем занятии. Это две функции, которые позволяют выжигать точку и передвигать рабочий инструмент (с включенным или отключенным лазером).

```
84 def laser_point (power, time):
85         """Выжигает отверстие, находясь в текущем положении рабочего инструмента"""
86         dType.SetEndEffectorLaser(api, 1, power, 1)
87         dType.dSleep(time)
88         dType.SetEndEffectorLaser(api, 0, power, 1)
89
90 def laser_line (power, x, y, v = 20, a = 20):
91         """Выжигает линию от текущей координаты до координаты x y"""
92         dType.SetPTPCommonParamsEx(api, v, a, 1)
93         if power == 0:
94                 a = 0
95         else:
96                 a = 1
97         dType.SetEndEffectorLaser(api, a, power, 1)
98         dType.SetPTPCmdEx(api, 2, x, y, 0, 0, 1)
99         dType.SetEndEffectorLaser(api, 0, 0, 1)
100
101 CON_STR = {
102     dType.DobotConnect.DobotConnect_NoError:  "DobotConnect_NoError",
103     dType.DobotConnect.DobotConnect_NotFound: "DobotConnect_NotFound",
104     dType.DobotConnect.DobotConnect_Occupied: "DobotConnect_Occupied"}
105
106 #load dll
107 api = dType.load()
108
109 #Connect Dobot
110 state = dType.ConnectDobot(api, "", 115200)[0]
111 print("Connect status:",CON_STR[state])
112
113 if (state == dType.DobotConnect.DobotConnect_NoError):
114
115         #Clean Command Queued
116         dType.SetQueuedCmdClear(api)
117
118         #Async Motion Params Setting
119         dType.SetHOMEParams(api, 200, 0, 0, 0, isQueued = 0)
120         dType.SetHOMECmd(api, 0, isQueued=1)
121         dType.SetPTPCommonParamsEx(api,20,20,1)
```

Рисунок VIII-8. Функции выжигания отверстия и перемещения рабочего инструмента с выжиганием и без него

Основная программа представлена на рисунке VIII-9. В бесконечном цикле сначала выводится основное меню — при помощи функции ***main_menu()*** в 127 строке. Затем пользователю предоставляется выбор из 4 действий. Если выбрано действие 1, то в 134 строке осуществляется вызов функции ***one_whole ()***, которая, как было показано выше, осуществляет опрос о параметрах выжигания одного отверстия. Затем выводится на экран информация об этом (строка 135). В строке 136 осуществляется выжигание.

Для пунктов 2 и 3 необходимо провести опрос о координатах для перемещения, а также о скорости и ускорении перемещения. Отдельно для пункта 3 необходимо узнать об интенсивности воздействия лазера.

В строках 150–153 вызывается функция, которая перемещает инструмент с выжиганием или без него.

Рисунок VIII-9. Основная часть программы для управления работы Dobot с клавиатуры

Если в 128 строке выбран пункт 4, то в 155 строке будет осуществлено прерывание цикла.

Задание для самостоятельной работы: осуществить рисование при помощи полученной программы с пользовательским интерфейсом.

ЗАДАНИЕ IX. Линейная функция

В дальнейшем мы рассмотрим, каким образом можно построить различные геометрические фигуры, такие как треугольник, прямоугольник или окружность. Часто бывает, что форма фигуры описывается какой-либо математической функцией. Математическая функция — это отображение одного множества другим таким образом, что элементу первого множества ставится в соответствие только один элемент второго множества.

Другими словами, функция ***f*** отображает множество ***x*** в множество ***y***. Функцию обозначают также записью $y = f(x)$, где ***x*** мы будем называть аргументом функции, а ***y*** — ее значением.

Часто под термином «функция» понимается числовая функция, то есть функция, которая ставит одни числа в соответствие другим. Эти функции удобно представлять в виде графиков. Именно построением графиков мы и займемся на протяжении этого и нескольких следующих заданий.

Самая простая функция — это так называемая линейная функция. Из названия становится ясным, что такая функция описывает построение линии на графике. В линейной функции осуществлена пропорциональная зависимость. То есть приращение аргумента функции пропорционально приращению значения функции. Линейная функция описывается следующей формулой:

$$y(x) = k \cdot x + b.$$

Эта запись означает, что ***y*** зависит от ***x*** — левая часть уравнения, а способ нахождения ***y*** при известном ***x*** — правая часть уравнения. В формуле в правой части уравнения присутствуют коэффициенты ***k*** и ***b***.

Цель данного задания — построить при помощи Dobot несколько линейных функций, изменяя значения ***k*** и ***b***. Для того чтобы наши результаты были более наглядными, воспользуемся сеткой точек, подобно той, которую мы получили на рисунке IV-4 когда создавали программу для выжигания сетки точек с интервалом в 10 мм. Но в этот раз воспользуемся функциями, которые мы разбирали в Задании VII («Функции в Python»).

```
import threading
import DobotDllType as dType

def laser_point (power, time):
        """Выжигает отверстие в текущем положение рабочего инструмента"""
        dType.SetEndEffectorLaser(api, 1, power, 1)
        dType.dSleep(time)
        dType.SetEndEffectorLaser(api, 0, power, 1)

def laser_line (power, x, y, v = 20, a = 20):
        dType.SetPTPCommonParamsEx(api, v, a, 1)
        if power == 0:
                s = 0
        else:
                s = 1
        dType.SetEndEffectorLaser(api, s, power, 1)
        dType.SetPTPCmdEx(api, 2, x, y, 0, 0, 1)
        dType.SetEndEffectorLaser(api, 0, 0, 1)

CON_STR = {
    dType.DobotConnect.DobotConnect_NoError:  "DobotConnect_NoError",
    dType.DobotConnect.DobotConnect_NotFound: "DobotConnect_NotFound",
    dType.DobotConnect.DobotConnect_Occupied: "DobotConnect_Occupied"}
#Load Dll
api = dType.load()
#Connect Dobot
state = dType.ConnectDobot(api, "", 115200)[0]
print("Connect status:",CON_STR[state])
if (state == dType.DobotConnect.DobotConnect_NoError):
        #Clean Command Queued
        dType.SetQueuedCmdClear(api)
        #Async Motion Params Setting
        dType.SetHOMEParams(api, 250, 0, 0, 0, isQueued = 0)
        dType.SetHOMECmd(api, 0, isQueued=1)
        dType.SetPTPCommonParamsEx(api,20,20,1)
print ("START")
for x in range (122, 323, 10):
        for y in range(-100,101,10):
                laser_line (0,x,y)
                print (x,y)
                laser_point (100,100)
print ("FINISH")
```

Рисунок IX-1. Выжигание сетки точек при помощи функций ***laser_point()*** *и* ***laser_line()***

В строках 4–16 нашего кода находятся упомянутые выше функции. В строках с 31 по 41 осуществляется перебор всех пар точек ***x, y*** в интервалах от 122 до 322 и от –100 до 100 соответственно.

Для реализации линейной зависимости необходимо написать код, представленный на рисунке IX-2. Линейная функция с параметрами $k = 0$ и $b = 3$:

```
print ("START")
k   0
b   3
x   122
y = k*(x 222) + b
laser_line (0,x,y,20,20)
laser_line (100,x,y,3,3)
for x in range (122, 323, 10):
        y = k*(x 222) + b
        laser_line (100,x,y,3,3)
        print (x,y)
print ("FINISH")
```

Рисунок IX-2. Линейная функция с параметрами $k = 0$ *и* $b = 3$

В строке 38 задается значение для коэффициента ***k***, в 39 — для ***b***. Строки 40–43 необходимы для того, чтобы рабочий инструмент пришел из точки ***home*** в начальную точку так, чтобы это произошло как можно быстрее и при выключенном лазере.

В цикле (44–47 строках) сначала вычисляется значение ***y***. А затем осуществляется перемещение рабочего инструмента с выжиганием. Главная сложность связана с тем, что значения координаты ***x*** в Dobot находятся в интервале от ***122*** до ***322***, в связи с чем для вычисления координаты ***y***, которая находится в интервале от ***–100*** до ***100***, необходимо из ***x*** вычесть ***222***. Таким образом, центром координат для Dobot будет пара ***0, 222***.

Для коэффициентов ***k = 0*** и ***b = 3***, а также ***k = 0*** и ***b = 53*** получены функции, представленные на рисунке IX-3. У нас получились две прямые параллельные оси ***x***. Это легко объяснимо, поскольку при любом ***x*** значение ***y*** будет оставаться неизменным и равным коэффициенту ***b***.

Рисунок IX-3. Графики линейных функций с коэффициентами ***k = 0, b = 3*** *и* ***k = 0, b = 53***

Давайте изменим значения коэффициентов на ***k = 1, b = 0*** и посмотрим результат на рисунке IX-4. Обратите внимание на то, что кривая сменила наклон. При этом приращение ***x*** теперь равно приращению ***y***.

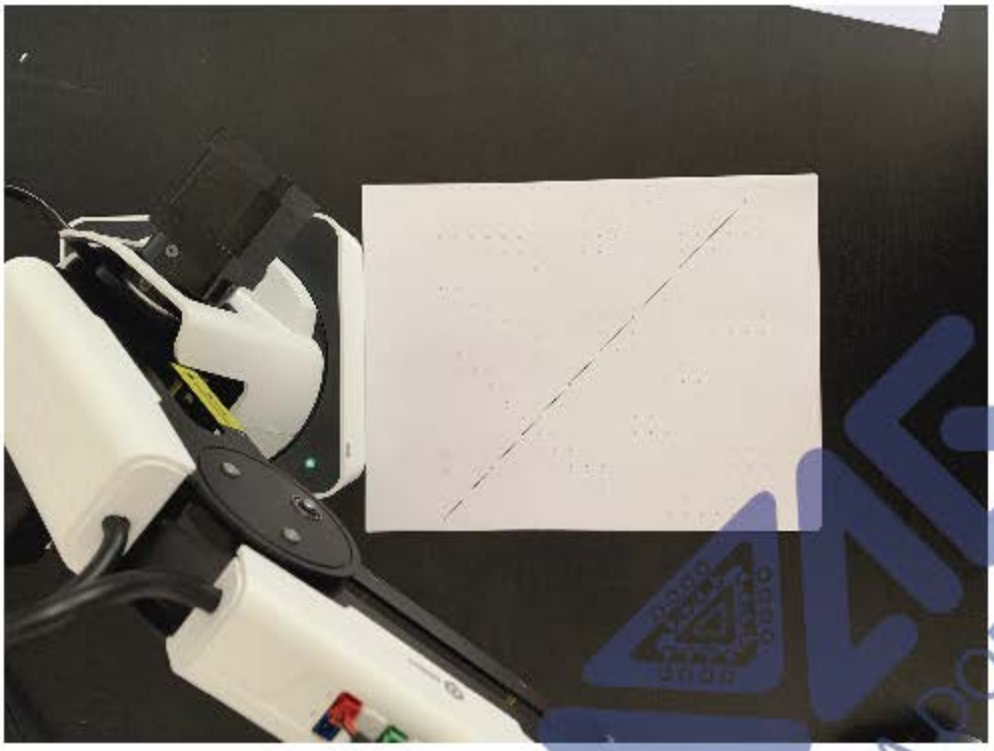


Рисунок IX-4. График линейной функции для коэффициентов $k = 1, b = 0$

Для того чтобы сделать наклон в другую сторону, необходимо изменить значение $k = 1$ на $k = -1$.

Рисунок IX-5. Линейные функции с коэффициентами $k = 1, b = 0$ *и* $k = -1, b = 0$

Задание для самостоятельной работы: модернизировать код для линейной функции таким образом, чтобы при выходе значений из размеров рабочей зоны Dobot не осуществлял печать.

ЗАДАНИЕ X. Линейная функция. Гипербола

Для построения линейной функции необходимо указать только две точки, а затем соединить их прямой линией. В дальнейшем мы рассмотрим и более сложные функции, но Dobot всегда будет перемещать свой рабочий инструмент от точки к точке по прямой. Если расстояние между точками будет достаточно маленьким, то возникнет иллюзия не ломаной, а именно кривой линии. В текущем задании нам предстоит выжечь серию линий таким образом, чтобы получить гиперболу (рисунок X-1).

Рисунок X-1. Выжигание при помощи Dobot серии линий, образующих гиперболу

Эта фигура получена рисованием серии отрезков. Начало отрезков всегда находится на линии, которую мы назовем ***ось x***, конец отрезков — на линии, параллельной этой оси (будем называть ее ***ось y***). Начальная точка первого отрезка находится на максимальном удалении от ***оси y***, конечная — на пресечении ***осей x*** и ***y***. Следующий отрезок начинается в точке, которая на 10 мм ближе к ***оси y***, заканчивается же он на 10 мм дальше от пересечения ***осей x*** и ***y***. Это справедливо для всех отрезков: начало всегда на ***оси x***, но при этом каждый следующий отрезок приближается к пересечению ***осей x*** и ***y*** на 10 мм относительно предыдущего, конец же отрезка всегда находится на ***оси y***, и при этом отдаляется от этого пересечения на 10 мм относительно предыдущего.

```
36 print ("START")
37 x = 322
38 laser line (0,x,100,20,20)
39 y   110
40 for x in range (322, 121, -10):
41             laser line (0,x,100,20,20)
42             y--10
43             laser_line (100,122,y,3,3)
44 print ("FINISH")
```

Рисунок X-2. Код для выжигания серии линий, образующих гиперболу

В коде на рисунке X-2. используется функция ***laser_line()***, которая была написана нами ранее. В 38 строке рабочий инструмент переходит из положения ***home*** в стартовую точку для первой прямой на ***оси x***. В цикле переменная ***x*** будет меняться с шагом в –10 от 322 до 122. Каждый раз для начала рисования рабочий инструмент будет перемещаться на ***ось x***, она находится на линии с координатой ***y*** = 100. Значение ***x*** при этом на каждой итерации будет уменьшаться на 10. Затем происходит уменьшение координаты ***y*** конечной точки отрезка и перемещение в эту точку на ***оси y*** в строке 43.

Если убрать все вырезанное из полученной фигуры, то получится гипербола, как это показано на рисунке X-3. Как можно видеть, прямые участки на ней практически не заметны.

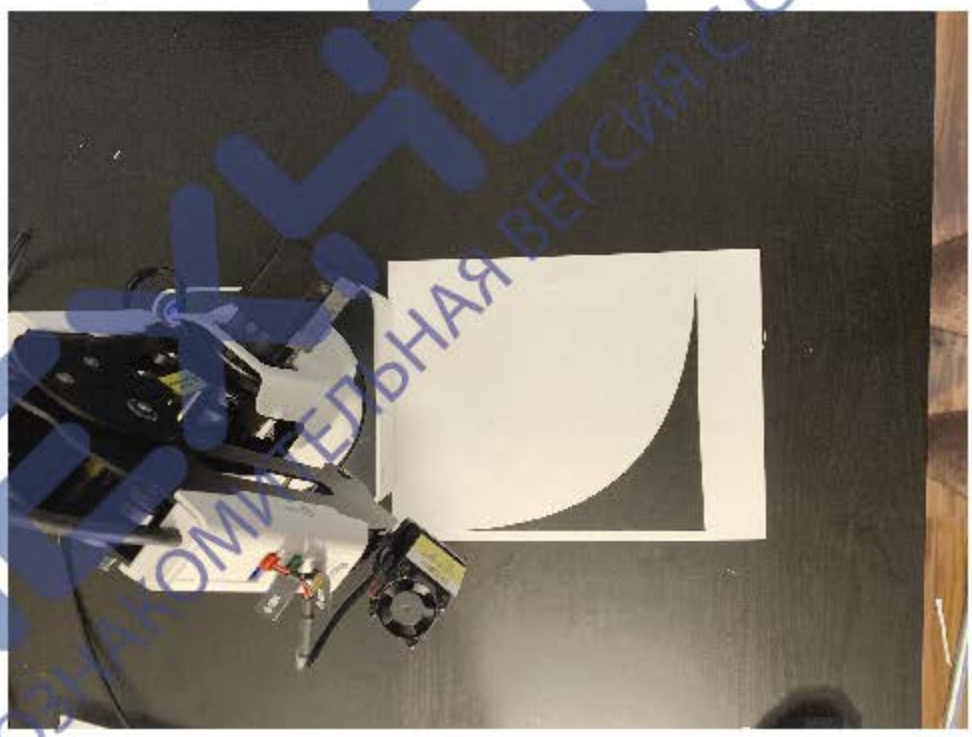


Рисунок X-3. Гипербола, вырезанная ***Dobot***

Код, изображенный на рисунке X-3 можно модернизировать, чтобы получить более сложную и интересную фигуру. Создадим 4 гиперболы, каждая из которых будет соприкасаться с соседними вертикальными и горизонтальными линиями. Как должен выглядеть код для этой программы, показано на рисунке X-4. Для каждой из гипербол создан свой цикл, для

каждого из цикла заданы свои стартовые и конечные точки для первой линии.

```
print ("START")

laser line (0,322,0,20,20)
y =  10
for x in range (322, 221,  10):
          laser line (0,x,0,20,20)
          y1=10
          laser_line (100,222,y,3,3)
laser_line (0,322,0,20,20)
y - 10

for x in range (322, 221,  10):
          laser_line (0,x,0,20,20)
          y-=10
          laser_line (100,222,y,3,3)

y - -10
for x in range (122, 223, 10):
          laser_line (0,x,0,20,20)
          y+ 10
          laser line (100,222,y,3,3)
laser line (0,322,0,20,20)

y   10
for x in range (122, 223, 10):
          laser_line (0,x,0,20,20)
          y -10
          laser_line (100,222,y,3,3)

print ("FINISH")
```

Рисунок X-4. Код для выжигания 4 гипербол

В результате получается удивительный по выразительности рисунок.

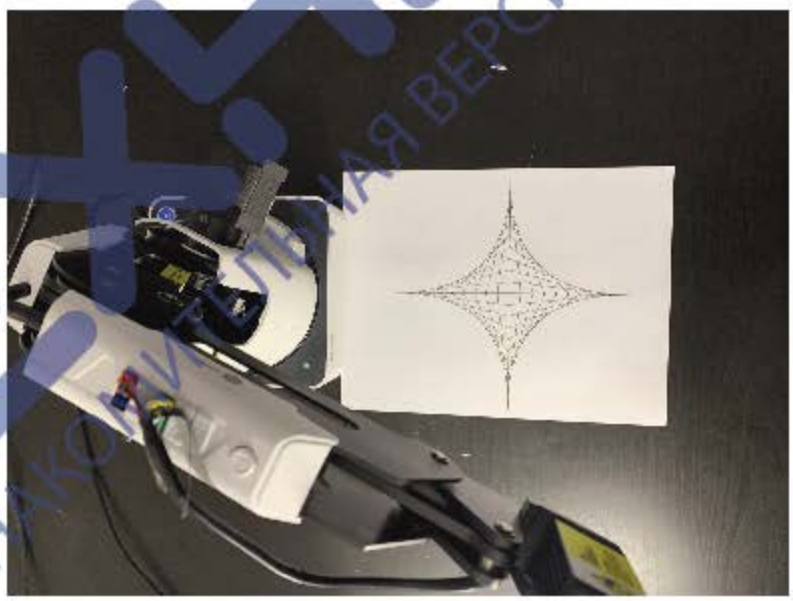


Рисунок X-5. Четыре гиперболы

Задание для самостоятельной работы: изменить шаги для рисования линий с 10 мм в большую или меньшую сторону; реализовать при помощи начертания линий другие функции.

ЗАДАНИЕ XI. Прямоугольник

На этом занятии мы продолжим изучать функции для рисования простых графиков. В будущем из этих графиков можно будет составить целый редактор для лазерного гравера.

Одной из простейших фигур является прямоугольник. Создадим функцию, которая позволит нарисовать прямоугольник со сторонами, параллельными ***осям x*** и ***y***. Существует несколько способов задать такой прямоугольник. Во-первых, можно указать все четыре вершины прямоугольника, но такой вариант является довольно подробным и может привести к ошибке, если пользователь или функция укажет неверное положение одной из вершин. Во-вторых, можно указать координаты одной из вершин, а затем задать длину и ширину. При этом необходимо договориться, что ширина — это отрезок, параллельный ***оси y***, а длина — отрезок, параллельный ***оси x***. Но в этом случае придется указывать направление рисования прямых от стартовых точек. В-третьих, можно указать центр прямоугольника (точка пересечения диагоналей) в качестве отправной точки для создания прямоугольника. Используем этот способ в функции прямоугольника — ***rectangle()***.

Для реализации функции ***rectangle()*** воспользуемся уже написанной нами функцией ***laser_line()***, как это показано на рисунке XI-1. Функция ***rectangle()*** принимает аргументы *x*, *y* — координаты центра (точки пересечения диагоналей прямоугольника), ***length*** и ***width*** — длину и ширину прямоугольника, а также *V* и ***a*** – скорость и ускорение движения рабочего инструмента, которые по умолчанию равны 3.

В 22 и 23 строках вычисляются координаты одной из вершин прямоугольника, а в 25 строке осуществляется перемещение рабочего инструмента к этой позиции при выключенном лазере.

В 26 строке при включенном лазере рабочий инструмент перемещается на длину прямоугольника — ***length***. Координата ***y*** при этом остается неизменной.

```
def laser_line (power, x, y, v = 20, a = 20):
        """Выжигает линию или перемещается из текущей координаты в координату (x,y)"""
        dType.SetPTPCommonParamsEx(api, v, a, 1)
        if power == 0:
                a = 0
        else:
                a = 1
        dType.SetEndEffectorLaser(api, a, power, 1)
        dType.SetPTPCmdEx(api, 2, x, y, 0, 0, 1)
        dType.SetEndEffectorLaser(api, 0, 0, 1)

def rectangle (x, y, length, width, v = 3, a = 3):
        x = x + length / 2
        y = y + width / 2
        print(x,y)
        laser_line (0, x, y)
        laser_line (100, x - length, y, v, a)
        print (x   length, y)
        laser_line (100, x - length, y - width, v, a)
        print (x - length, y + width)
        laser_line (100, x, y   width, v, a)
        print (x - length, y - width)
        laser_line (100, x, y, v, a)
        print (x, y)
```

*Рисунок XI-1. Функция **rectangle()** для выжигания прямоугольника*

В 28 строке осуществляется смещение по оси ***y*** на ширину ***width***.

В 29 строке Dobot снова перемещается только по ***оси x*** с неизменным перемещением по ***оси y.***

Последнее перемещение осуществляется в начальную точку.

Осуществим вызов функции ***rectangle()*** таким образом, чтобы в центре нашей рабочей зоны получился квадрат со стороной ***100 мм.*** Код для этой программы будет выглядеть следующим образом (рисунок XI-2).

```
def rectangle (x, y, length, width, v = 3, a = 3):
        x = x + length / 2
        y = y + width / 2
        print(x,y)
        laser_line (0, x, y)
        laser_line (100, x - length, y, v, a)
        print (x   length, y)
        laser_line (100, x - length, y - width, v, a)
        print (x - length, y + width)
        laser_line (100, x, y   width, v, a)
        print (x   length, y   width)
        laser_line (100, x, y, v, a)
        print (x, y)


CON_STR = {
    dType.DobotConnect.DobotConnect_NoError:  "DobotConnect_NoError",
    dType.DobotConnect.DobotConnect_NotFound: "DobotConnect_NotFound",
    dType.DobotConnect.DobotConnect_Occupied: "DobotConnect_Occupied"}
#Load Dll
api = dType.load()
#Connect Dobot
state = dType.ConnectDobot(api, "", 115200)[0]
print("Connect status:",CON_STR[state])
if (state == dType.DobotConnect.DobotConnect_NoError):
        #Clean Command Queued
        dType.SetQueuedCmdClear(api)
        #Async Motion Params Setting
        dType.SetHOMEParams(api, 250, 0, 0, 0, isQueued = 0)
        dType.SetHOMECmd(api, 0, isQueued=1)
        dType.SetPTPCommonParamsEx(api,20,20,1)
print ("START")

rectangle (222, 0, 100, 100)


print ("FINISH")
```

*Рисунок XI-2. Вызов функции **rectangle()** с аргументами 222, 0, 100, 100*

После вызова функции ***rectangle ()*** и ее запуска будет получен следующий результат:

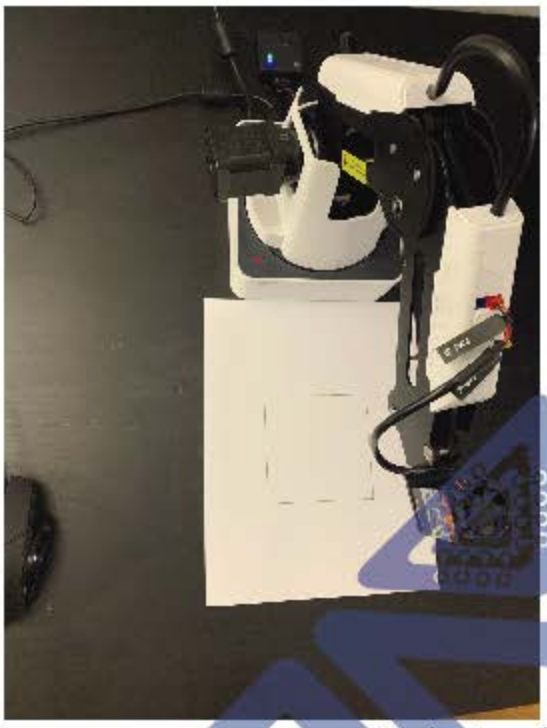


Рисунок XI-3. Выжигание при помощи Dobot квадрата в центре рабочей зоны

Давайте усложним задачу и создадим серию увеличивающихся прямоугольников с общим центром, как на рисунке XI-4.

Рисунок XI-4. Концентрические прямоугольники, выжженные при помощи Dobot

Код для реализации выжигания серии концентрических прямоугольников представлен на рисунке XI-5. В данном случае при каждом шаге в 10 мм изменяется и длина прямоугольника от 10 до 100 мм, ширина же каждого прямоугольника составляет половину его длины.

```
print ("START")

for length in range (10,110,10):
            rectangle (222, 0, length, length/2)


print ("FINISH")
```

Рисунок XI-5. Код для реализации серии концентрических прямоугольников

Задание для самостоятельной работы: разработать рисунки и орнаменты на основе прямоугольников, реализовать выжигание этих объектов.

ЗАДАНИЕ XII. Изображение цифр при помощи прямоугольников

В этом задании нам необходимо будет создать упрощенный шрифт для того, чтобы Dobot, основываясь на этом шрифте, мог выжигать цифры. Наш шрифт будет подобен почтовому индексу, который используется для при отправлении писем и посылок (рисунок XII-1).

Рисунок XII-1. Написание почтового индекса

В нашем случае шрифт будет вписан в прямоугольник с шириной 4 и длиной 8 квадратов.

Кроме того, нам понадобятся функции, которые мы создали на предыдущих занятиях: ***rectangle()*** и ***laser_line()***. Каждая цифра будет состоять из серии квадратов. Те квадраты, которые необходимо выжечь, будут записаны в список Python как ***1***, а те, которые должны остаться пустыми, — как ***0***.

Например, на рисунке XII-2 можно увидеть массив для выжигания цифры 2. Обратите внимание, что получился обычный одномерный массив, для удобства и простоты чтения он разбит на несколько строк. При этом, если приглядеться, можно разглядеть цифру 2, написанную единичками.

```
53 n2 = [0, 1, 1, 0,
54       1, 0, 0, 1,
55       1, 0, 0, 1,
56       0, 0, 1, 0,
57       0, 1, 0, 0,
58       1, 0, 0, 0,
59       1, 0, 0, 1,
60       1, 1, 1, 1]
61
```

Рисунок XII-2. Список для выжигания при помощи Dobot цифры 2

Для того чтобы выжечь эту цифру квадратами, необходимо реализовать следующий код (рисунок XII-3).

```
n2 = [0, 1, 1, 0,
      1, 0, 0, 1,
      1, 0, 0, 1,
      0, 0, 1, 0,
      0, 1, 0, 0,
      1, 0, 0, 0,
      1, 0, 0, 1,
      1, 1, 1, 1]

print ("START")
i=0
for x in range (142,302,20):
          for y in range (-30, 50, 20):
                    if n2 [i] ==1:
                              rectangle (x, y, 20, 20)
                    i+=1
print ("FINISH")
```

Рисунок XII-3. Код для выжигания цифры 2 квадратами

В коде используется вложенный цикл, чтобы перебрать все пары координат ***x*** и ***y***. Ребро каждого квадрата, из которых состоит каждый из прямоугольников, составляет ***20 мм***. В строке 66 осуществляется проверка элемента с индексом ***i*** списка n2. Если он равен 0, то квадрат выжигается.

Рисунок XII-4. Цифра 2, выжженная при помощи Dobot

Код, представленный на рисунке XII-3 можно модернизировать для выжигания цифр других размеров и иного написания. Для этого необходимо

создать списки выжигания для каждой цифры. Удобнее всего для решения этой задачи воспользоваться обычным тетрадным листом в клетку и карандашом. Именно так мы и создали свои шаблоны цифр (рисунок XII-5).

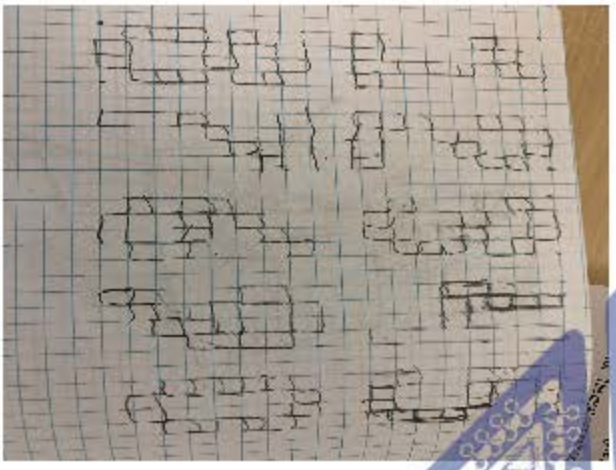


Рисунок XII-5. Шаблоны цифр

Чтобы выжечь цифры согласно составленным образцам, каждый из этих шаблонов необходимо записать в отдельную функцию, аргументом у которой будет цифра, а возвращаемым — список для выжигания этой цифры.

```
47 def numbers (n):
48             n0    [0, 1, 1, 0,
49                    1, 0, 0, 1,
50                    1, 0, 0, 1,
51                    1, 0, 0, 1,
52                    1, 0, 0, 1,
53                    1, 0, 0, 1,
54                    1, 0, 0, 1,
55                    0, 1, 1, 0]
56
57             n1 = [0, 0, 1, 0,
58                   0, 0, 1, 0,
59                   0, 1, 1, 0,
60                   1, 0, 1, 0,
61                   0, 0, 1, 0,
62                   0, 0, 1, 0,
63                   0, 0, 1, 0,
64                   1, 1, 1, 1]
65
66             n2 = [0, 1, 1, 0,
67                   1, 0, 0, 1,
68                   1, 0, 0, 1,
69                   0, 0, 1, 0,
70                   0, 1, 0, 0,
71                   1, 0, 0, 0,
72                   1, 0, 0, 1,
73                   1, 1, 1, 1]
74
75             n3 = [0, 1, 1, 0,
76                   1, 0, 0, 1,
77                   0, 0, 0, 1,
78                   0, 1, 1, 0,
79                   0, 0, 0, 1,
80                   1, 0, 0, 1,
81                   1, 0, 0, 1,
82                   0, 1, 1, 0]
```

Рисунок XII-6. Списки для выжигания цифр от 0 до 3

```
84              n4 = [0, 1, 0, 1,
85                    0, 1, 0, 1,
86                    0, 1, 0, 1,
87                    1, 0, 0, 1,
88                    1, 1, 1, 1,
89                    0, 0, 0, 1,
90                    0, 0, 0, 1,
91                    0, 0, 0, 1]
92
93              n5 = [1, 1, 1, 1,
94                    1, 0, 0, 0,
95                    1, 0, 0, 0,
96                    1, 1, 1, 0,
97                    0, 0, 0, 1,
98                    0, 0, 0, 1,
99                    1, 0, 0, 1,
100                   0, 1, 1, 0]
101
102             n6 = [0, 0, 0, 1,
103                   0, 0, 1, 0,
104                   0, 1, 0, 0,
105                   1, 1, 1, 0,
106                   1, 0, 0, 1,
107                   1, 0, 0, 1,
108                   1, 0, 0, 1,
109                   0, 1, 1, 0]
110
111             n7 = [1, 1, 1, 1,
112                   0, 0, 0, 1,
113                   0, 0, 1, 0,
114                   0, 1, 0, 0,
115                   1, 0, 0, 0,
116                   1, 0, 0, 0,
117                   1, 0, 0, 0,
118                   1, 0, 0, 0]
```

Рисунок XII-7. Списки для выжигания цифр от 4 до 7

```
120             n8 = [0, 1, 1, 0,
121                   1, 0, 0, 1,
122                   1, 0, 0, 1,
123                   0, 1, 1, 0,
124                   1, 0, 0, 1,
125                   1, 0, 0, 1,
126                   1, 0, 0, 1,
127                   0, 1, 1, 0]
128
129             n9 = [0, 1, 1, 0,
130                   1, 0, 0, 1,
131                   1, 0, 0, 1,
132                   0, 1, 1, 1,
133                   0, 0, 0, 1,
134                   0, 0, 1, 0,
135                   0, 1, 0, 0,
136                   1, 0, 0, 0]
137
138
139             num = [n0, n1, n2, n3, n4, n5, n6, n7, n8, n9]
140             return num[n]
141
```

Рисунок XII-8. Списки для выжигания цифр 8 и 9

Кроме того, необходимо написать функцию ***numbers_laser()*** для вырезания каждой из цифр, как это показано на рисунке XII-9.

```
142 def numbers_laser (num, x,y, size):
143
144             i=0
145             print (num)
146             n = numbers (num)
147             for xt in range (x,x + size * 8, size):
148                     for yt in range (y, y + size *4, size):
149                             if n [i] ==1:
150                                     rectangle (xt, yt, size, size)
151                             i+=1
152
153 print ("START")
154 for i in range (0, 10):
155             numbers_laser (i, 150,   80 + i * 12, 2)
156 print ("FINISH")
157
```

*Рисунок XII-9. Функция **numbers_laser()** для выжигания символов*

Данная функция принимает значение ***num***, которое затем передается функции ***numbers()*** для получения списка ***n*** со значениями для выжигания цифры ***num***. Кроме того, аргументами функции ***numbers_laser()*** выступают координаты ***x*** и ***y*** верхнего левого прямоугольника цифры, а также ***size*** — размер каждого из квадратов, из которых состоит цифра.

Если запустить данную функцию с параметрами

numbers_laser(5, 222, 0,5),

результат будет выглядеть так же, как на рисунке XII-10.

Рисунок XII-10. Цифра 5, выжженная при помощи Dobot

Если же выполнить код в строках 154, 155, будут выжжены все цифры, при этом длина стороны каждого квадрата будет 2 мм.

Рисунок XII-11. Цифры, выжженные при помощи Dobot

Задание для самостоятельной работы: создать шаблоны для выжигания букв: вписать их в функцию ***numbers()***.

ЗАДАНИЕ XIII. Многоугольники

В предыдущих заданиях ребра наших фигур были параллельны или перпендикулярны координатным осям. Однако этого функционала недостаточно для создания цельного графического редактора. В этом задании мы научимся создавать функцию, которая позволит рисовать многоугольники, опираясь на их вершины. То есть функция, получая на входе координаты вершин, будет выжигать многоугольник, последовательно прожигая прямые линии, которые соединяют соседние вершины. Необходимо также замкнуть первую и последнюю вершины.

```
31 def polygon (vertex):
32             """Выжигает многоугольник с вершинами из списка списков vertex"""
33             laser_line (0, vertex[0][0], vertex[0][1])
34             for v in vertex[1:]:
35                         laser_line (100, v[0], v[1], 3, 3)
36                         print (v[0], v[1])
37             laser_line (100, vertex[0][0], vertex[0][1],3,3)
```

Рисунок XIII-1. Функция для выжигания многоугольников

Обратите внимание на то, что аргумент ***vertex*** — это список списков. То есть это список точек с двумя координатами: ***x*** и ***y***. В 33 строке используется написанная нами ранее функция ***laser_line()***. По умолчанию она передвигает рабочий инструмент со скоростью и ускорением ***20***. В 33 строке он передвигается с выключенным лазером. В 34 строке обьявляется цикл, который по мере реализации будет присваивать переменной ***v*** значения из ***vertex*** (все, кроме нулевого). Переменная ***v*** — это список из двух элементов, которые представляют собой координаты текущей точки. В цикле перебор начинается со второй точки, а движение — с первой, так как в первую точку рабочий инструмент попадает еще в 33 строке, при выключенном лазере. В 37 строке многоугольник замыкается — осуществляется перемещение в первую точку.

Давайте создадим этот код для нескольких многоугольников. Начнем с треугольника (рисунок XIII-2).

```
93 print ("START")
94 vertex = [[200,-50], [200, 50], [250,0]]
95 polygon(vertex)
96 print ("FINISH")
```

Рисунок XIII-2. Выжигание треугольника по вершинам

В 94 строке координаты вершин треугольников задаются попарно, они передаются в функцию ***polygon(),*** код которой написан выше. В результате получается равнобедренный треугольник, как на рисунке XIII-3.

Рисунок XIII-3. Равнобедренный треугольник, выжженный при помощи Dobot

Теперь создадим ромб. Для этого необходимо между первой и второй точкой добавить еще одну, у которой координата *y* будет равна третьей координате треугольника, а координата *x* будет равна координате *x* первой точки минус разница координат *x* второй и третьей точек.

```
93 print ("START")
94 vertex = [[200,-50], [150, 0], [200, 50], [250,0]]
95 polygon(vertex)
96 print ("FINISH")
```

Рисунок XIII-4. Код для выжигания ромба

Рисунок XIII-5. Ромб, выжженный при помощи Dobot

Код для выжигания шестиугольника выглядит следующим образом (рисунок XIII-6).

```
print ("START")
x = 300
y =  80
vertex = [[x,y], [x+5, y+5], [x+5, y+5+10], [x,y+5+10+5], [x-5, y+10+5], [x-5,y+5]]
polygon(vertex)
print ("FINISH")
```

Рисунок XIII-6. Код для выжигания шестиугольника

Данный код можно модернизировать, чтобы получить орнамент из нескольких шестиугольников.

```
print ("START")
for x in range (300, 320,  10):
        for y in range ( 80,100, 20):
                vertex = [[x,y], [x+5, y+5], [x+5, y+5+10], [x,y+5+10+5], [x-5, y+10+5], [x-5,y+5]]
                polygon(vertex)
print ("FINISH")
```

Рисунок XIII-7. Код для выжигания орнамента из нескольких шестиугольников

Для этого необходимо во вложенных циклах изменять координаты ***x*** и ***y***, которые характеризуют положение каждого из шестиугольников.

Рисунок XIII-8. Орнамент из серии шестиугольников

Задание для самостоятельной работы: создать собственные фигуры для выжигания повторяющихся орнаментов, например треугольники, ромбы или равнобедренные трапеции.

ЗАДАНИЕ XIV. Треугольник. Тригонометрические функции. Выжигание треугольника по двум углам и стороне между ними

Базовой фигурой для построения любого графического редактора, в том числе и того, который программирует поведение Dobot, является треугольник. И для этого есть несколько причин. Одна из наиважнейших — то, что любую фигуру можно рассматривать в качестве набора треугольников. Например, любой выпуклый четырехугольник можно представить в виде двух треугольников, если провести диагональ. Даже круг можно представить в виде множества равнобедренных треугольников с бесконечно малым основанием и боковыми сторонами, равными радиусу окружности.

Но есть еще одно важное свойство треугольников. В любом треугольнике можно провести высоту — линию, которая соединяет вершину и противоположную сторону и является перпендикуляром к этой стороне. В этом случае у нас получается два прямоугольных треугольника. Для каждого из этих треугольников можно применить теорему Пифагора, которая связывает длины всех трех сторон соотношением:

$$\textbf{\textit{Гипотенуза}}^2 = \textbf{\textit{Катет1}}^2 + \textbf{\textit{Катет2}}^2.$$

Давайте нарисуем несколько таких треугольников. При этом один из катетов у них будет находиться на ***оси x***, следовательно, второй катет будет параллелен ***оси y***. Кроме того, у всех треугольников будет одна общая вершина, которая находится на пересечении катета, параллельного ***оси x***, и гипотенузы. Каждая сторона последующего треугольника должна быть больше, чем предыдущая, во столько раз, во сколько и любая другая сторона. Код для данной задачи показан на рисунке XIV-1.

```
93 print ("START")
94
95 for k in range (1, 6):
96         vertex = [[152, 80],[(k*20+152),( 80)],[(k*20+152),(k*20 80)]]
97         polygon(vertex)
98
99 print ("FINISH")
```

Рисунок XIV-1. Код для выжигания серии подобных треугольников

В 95 строке реализован цикл с итерируемой переменной ***k***, которая примет значения от 1 до 5 включительно. В 96 строке задаются параметры треугольников. Обратите внимание, что длина катетов зависит от числа ***k***.

Результат выполнения программы можно увидеть на рисунке XIV-2. Можно легко убедиться, что у этих треугольников одинаковые углы, а соотношение сторон подчиняется следующей закономерности:

$$\frac{a}{A} = \frac{b}{B} = \frac{c}{C},$$

где строчные буквы характеризуют один треугольник, а прописные — другой. В нашем случае у «соседних» треугольников эта дробь будет равна 20, как следует из кода, представленного выше. Из формулы также видно, что при увеличении или уменьшении треугольников неизменным остается соотношение сторон в каждом из треугольников:

$$\frac{a}{b}=\frac{A}{B}$$

Так как во всех треугольниках неизменными остаются углы, очень удобно соотнести полученные соотношения сторон с величинами углов. Эти соотношения называются тригонометрическими функциями: ***sin*** (синус угла треугольника), ***cos*** (косинус угла треугольника) и ***tg*** (тангенс угла треугольника).

Рисунок XIV-2. Серия подобных треугольников

$$\sin(\text{угла треугольника}) = \frac{\text{противолежащий катет}}{\text{гипотенуза}};$$

$$\cos(\text{угла треугольника}) = \frac{\text{прилежащий катет}}{\text{гипотенуза}};$$

$$\operatorname{tg}(\text{угла треугольника}) = \frac{\text{противолежащий катет}}{\text{прилежащий катет}}.$$

Важно помнить, что данные соотношения справедливы только для прямоугольных треугольников и являются по сути следствием теоремы Пифагора. Однако, как указывалось выше, любой из треугольников можно рассматривать как сумму двух прямоугольных треугольников.

В целом для любых треугольников действуют две теоремы, которые позволяют полностью определить треугольник, если нам известны некоторые из его параметров, такие как углы, которые его образуют, или стороны.

Первая теорема — это теорема синусов (где ***a***, ***b***, ***c*** — стороны треугольника):

$$\frac{a}{\sin(\text{угла против } a)} = \frac{b}{\sin(\text{угла против } b)} = \frac{c}{\sin(\text{угла против } c)}.$$

Обратите внимание, что достаточно знать любые две стороны и угол между ними или все три стороны, или любую сторону и два угла, между которыми она находится, чтобы полностью восстановить все стороны и углы треугольника.

Вторая теорема — это теорема косинусов:

$$c^2 = a^2 + b^2 + 2ab\cos(\text{угла между } a \text{ и } b).$$

Обе теоремы являются прямым следствием теоремы Пифагора.

Для построения нашего графического редактора очень удобно использовать функцию, которая будет получать в качестве аргументов одну из сторон (точнее, координаты ее концов) и два угла, между которыми она находится, а возвращать — координаты всех вершин треугольника. Впоследствии эти вершины можно передать в функцию ***polygon()*** для выжигания такого треугольника.

Для того чтобы осуществить подобную операцию, необходимо подключить библиотеку ***math***, которая позволяет рассчитывать значения тригонометрических функций. Для этого следует ввести код:

```
import math.
```

После завершения всех манипуляций написанная функция будет выглядеть следующим образом (рисунок XIV-3).

```
77 def triangle (a, alpha, betta):
78         '''Находит координаты трех вершин треугольника,
79         при этом первая вершина находится в точке (0,0), вторая на оси '''
80         c = a * math.sin(betta * math.pi / 180) / math.sin ((alpha + betta) * math.pi / 180)
81         x2 = c * math.cos(alpha * math.pi / 180)
82         y2 = c * math.sin(alpha * math.pi / 180)
83         return [[0,0],[a,0],[x2,y2]]
```

*Рисунок XIV-3. Функция **triangle()** для расчета вершин треугольника по двум углам и стороне между ними*

Функция ***triangle()*** принимает три аргумента, как это видно из строки 77. В 80 строке осуществляется расчет одной из сторон — ***c***. Все эти построения отражены на рисунке XIV-4, демонстрирующем нахождение координат треугольника из двух углов и ребра между ними. Все расчеты производятся на основе теоремы косинусов. Затем, зная длину ***c*** и опираясь на определения косинуса и синуса, мы можем легко рассчитать значения x_2 и y_2.

Важно помнить, что для тригонометрических функций в Python используются значения углов в радианах, а не в градусах. Для перевода градусов в радианы значение каждого из углов делится на ***180°***, а затем умножается на ***π***. Функция ***triangle()*** возвращает значение трех вершин треугольника.

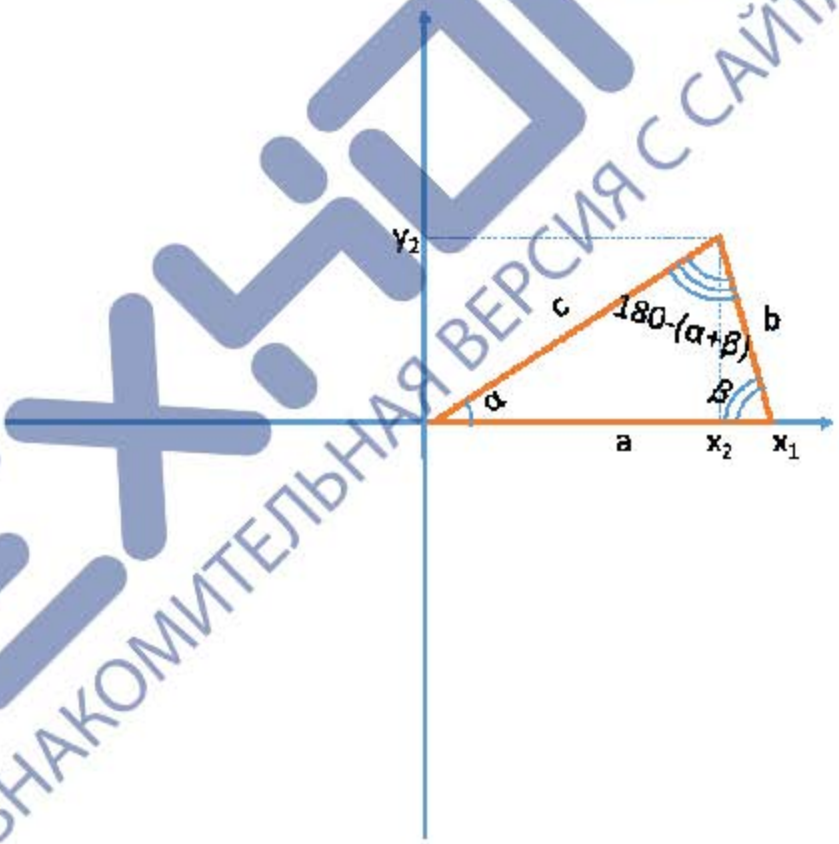


Рисунок XIV-4. Нахождение координат треугольника из двух углов и ребра между ними

Давайте рассмотрим различные примеры работы полученной функции.

```
103 print ("START")
104
105 print (triangle (100, 45, 45))
106 a = triangle (100, 30, 60)
107 vertex = [[i[0]+122, i[1]-100] for i in a]
108 print (vertex)
109 polygon (vertex)
110
111 print ("FINISH")
```

Рисунок XIV-5. Расчет и печать равнобедренного прямоугольного треугольника

Введем в качестве углов значения 45 градусов. Не забудем также изменить координаты каждой из вершин треугольника на 122 — *x* и на –100 — *y*, как это показано в строчке 107.

Рисунок XIV-6. Равнобедренный прямоугольный треугольник

Изменим углы при гипотенузе на 30° и 60°. Нами будет получен прямоугольный треугольник, у которого катет, лежащий против угла в 30°, будет равен половине гипотенузы, в чем мы сможем легко убедиться, если используем обыкновенную линейку (рисунок XIV-7).

Рисунок XIV-7. Прямоугольный треугольник с углом 30°

Последним тестовым треугольником будет равносторонний треугольник, у которого все углы равны 60°. Для его построения нам достаточно ввести одну сторону и один угол (рисунок XIV-8).

Рисунок XIV-8. Равносторонний треугольник

На рисунке XIV-9 приведен пример, в котором осуществляется построение восьми прямоугольных треугольников с общим основанием. Каждый последующий треугольник отличается от предыдущего тем, что один из углов у гипотенузы возрастает на 10°, а второй соответственно уменьшается на 10°.

```
103 print ("START")
104
105 for alp in range (10, 90, 10):
106             a = triangle (100, alp, 90   alp)
107             vertex = [[i[0]+122, i[1]-100] for i in a]
108             polygon (vertex)
109
110 print ("FINISH")
```

Рисунок XIV-9. Код, иллюстрирующий построение вписанных в окружность углов

В результате выполнения этого кода получится любопытное построение (рисунок XIV-10). На рисунке хорошо видно, что вершины треугольников, равные 90°, образуют контур окружности. Этот угол всегда равен половине дуги окружности, на которую опирается треугольник в нашем случае это 180°.

*Рисунок XIV-10. Вписанные в окружность углы в **90°***

Задание для самостоятельной работы: реализовать функции, которые будут возвращать координаты треугольников по введенным трем сторонам или по двум сторонам и углу между ними.

ЗАДАНИЕ XV. Тригонометрические функции

В прошлом задании нами были использованы тригонометрические функции для того, чтобы найти все параметры треугольников. Как было сказано, треугольники интересны прежде всего тем, что любую фигуру можно представить в качестве суммы треугольников. Именно поэтому нам предстоит более подробно изучить свойства тригонометрических функций.

Для начала необходимо разобраться, что происходит с синусом или косинусом при увеличении или уменьшении соответствующего угла. Построим (а вернее, выжжем) график зависимости синуса и косинуса от угла. Учтем также, что нам необходим максимально возможный размер рабочей области. Для того чтобы найти наиболее удобное соотношение длины и ширины, вернемся к рисунку II-12. Нам необходимо подобрать такую область, которая имела бы максимальную ширину (разброс по ***оси y***) и была в длину около ***100 мм***. Исходя из этого требования мы и выбираем диапазон значений по ***x*** [122, 222] и по ***y*** [–270, 270]. Для удобства нам следует вырезать при помощи Dobot на одном листе две зависимости: ***y = sin (x)*** и ***y = cos (x)***. А также ***ось x***, ***ось y*** и прямые, параллельные ***оси y***, в которых интересно будет проследить изменение этих функций. Кроме того, чтобы рассмотреть как можно больше значений, мы будем вынуждены поменять ***оси y*** и ***x*** наших функций местами. То есть ***ось y*** Dobot будет соответствовать ***x*** функции ***y = sin x*** (рисунок XV-1).

```
103 print ("START")
104
105 laser_line (0, 172, 270)
106 laser_line (1, 172, 270, 4,4)
107 for i in range ( 270,272, 80):
108             laser_line (0, 122, i)
109             laser_line (1, 222, i, 4,4)
110 sin = [[x , 172 + 50*math.sin(x*math.pi/100)] for x in range(-270, 271, 2)]
111 cos = [[x , 172 + 50*math.cos(x*math.pi/180)] for x in range(-270, 271, 2)]
112
113 laser_line(0, sin[0][1], sin[0][0])
114
115 for i in sin:
116             laser_line(0, i[1], i[0],20,20)
117             laser_point(100,300)
118
119 laser_line(0, cos[0][1], cos[0][0])
120
121 for i in cos:
122             laser_line(100, i[1], i[0],5,5)
123
124
125 print ("FINISH")
```

*Рисунок XV-1. Код для выжигания функций **sin** и **cos***

Как видно из кода, в 105 строке осуществляется перемещение инструмента без включения лазера для выжигания ***оси x*** (в данном случае ***ось x*** не совпадает с осью ***x*** Dobot, а совпадает с ***осью y***), затем в строке 106 про-

изводится выжигание этой оси. В данном задании ось **x** проходит между координатами (172; –270) и (172; 270).

В цикле строк 107–109 осуществляется выжигание вертикальных линий, параллельных ***оси y***.

В 110 строке при помощи генератора Python создается многомерный список ***sin***, который включает два списка: список для аргументов ***x***, которые находятся в промежутке –270, 270 с шагом в 2, и значений

172 + 50 *math.sin(x*math.pi/180).

Значение градусов при этом переводится в радианы, а максимальное отклонение графика функции от центральной линии, которая проходит через координату 172, равно 50.

То же происходит и в строке 111 для ***cos***.

Для того чтобы функции ***cos*** и ***sin*** различались между собой, на выжженном рисунке их необходимо нарисовать по-разному. Для этого в цикле в строках 115–118 осуществляется последовательное перемещение и выжигание точек для функции ***sin***.

В строках 121–122 осуществляется выжигание ***cos*** линиями.

Для более быстрого перемещения рабочего инструмента из последней выжженной точки ***sin*** к первой точке ***cos*** в 119 строке осуществляется перемещение рабочего инструмента с наибольшими скоростью и ускорением при выключенном лазере.

Результат выполнения кода можно увидеть на рисунке XV-2. Обратите внимание, что крайние кривые, параллельные ***оси y***, получились урезанными в связи с тем, что в этих зонах рабочий инструмент выходил за пределы рабочей области. Функция ***sin*** выжжена точками, а ***cos*** — сплошной линией. Обратите внимание и на то, что при наибольшем отклонении от центральной линии одной функции вторая равна нулю. При возрастании одной функции уменьшается другая, поскольку каждая из функций пропорциональна размеру одного из катетов треугольника, а при увеличении одного катета при неизменной гипотенузе уменьшается другой катет. Кроме того, мы можем заметить, что функции ***sin*** и ***cos*** являются периодическими, то есть повторяющимися через определенный промежуток. Так, значения ***sin*** равны ***0*** при — ***180°***, ***0°*** и ***180°***.

*Рисунок XV-2. Графики функций **sin** и **cos***

На рисунке XV-3 представлен код, который в 110 и 111 строках осуществляет генерирование двух многомерных списков. Затем в цикле в строке 117 осуществляется выжигание функции $sin^2 + cos^2$.

```
105 print ("START")
106
107 laser_line (0, 172, -90)
108 laser_line (1, 172, 90, 4,4)
109 for i in range( 90,91, 90):
110             laser_line (0, 122, 4)
111             laser_line (1, 222, 1, 4,4)
112 sin = [i, math.sin(x*math.pi/180)| for x in range(-90, 91, 2)]
113 cos = [ix , math.cos(x*math.pi/180)| for x in range( 90, 91, 2)]
114
115 laser_line(0, 172 + 50*(sin[0][1]*sin[0][1] + cos[0][1]*cos[0][1]), sin[i][0])
116
117 for i in range (0, 91, 1):
118
119             laser_line(100, 172 + 50*(sin[i][1]*sin[i][1] + cos[i][1]*cos[i][1]), sin[i][0] ,5, 5)
120
121
122 print ("FIN:50")
```

Рисунок XV-3. Код для выжигания функции $sin^2 + cos^2$

В данном коде диапазон аргументов от –90° до 90°. Выжженная линия показана на рисунке XV-4. Как видно, сумма квадратов синуса и косинуса — это всегда **1**. Для наглядности сумма квадратов синуса и косинуса умножена на 50 в 113 строке, а затем к этому значению прибавлено 172 (положение центра рабочей области Dobot).

Рисунок XV-4. Функция $\boldsymbol{sin^2 + cos^2}$

Задание для самостоятельной работы: создать программу для выжигания тригонометрической функции тангенса.

Важно! Учтите, что значение тангенса определено не на всем диапазоне: например, невозможно вычислить значение тангенса для угла в 0 градусов.

ЗАДАНИЕ XVI. Круг, дуга, сегмент, сектор

Настало время научиться создавать одну из важнейших линий — окружность. Частично мы уже решали эту задачу, когда создавали контур окружности при помощи серии прямоугольных треугольников (занятие XIV). Но сейчас создадим эту кривую классическим способом. Для этого нам понадобится вернуться к занятию XV и проанализировать последнюю полученную закономерность.

$$\boldsymbol{sin^2(alp) + cos^2(alp) = 1.}$$

Или другими словами:

$$\left(\frac{\text{катет1}}{\text{гипотенуза}}\right)^2 + \left(\frac{\text{катет2}}{\text{гипотенуза}}\right)^2 = 1,$$

$$\text{катет1}^2 + \text{катет2}^2 = \text{гипотенуза}^2.$$

Мы знаем, что все точки окружности лежат на одном расстоянии от ее центра, и это расстояние называется радиусом. Если представить, что центр окружности находится в центре системы координат, то можно провести множество прямых от центра к окружности. Их длина будет равна радиусу, а проекции этих прямых на ***осях x*** и ***y*** будут равны катетам прямоугольного треугольника. В этом случае для построения окружности необходимо для каждой точки рассчитывать величины только катетов проекций. Код функции для создания этого подхода показан на рисунке XVI-1. У этой функции 7 аргументов, но на данном этапе нам необходимы только 6 первых (поэтому седьмой аргумент по умолчанию равен 0):

1, 2) центр окружности, выражаемый координатами x_0 и y_0;

3) радиус окружности;

4) начальный угол для выжигания ***alp0*** относительно оси ***x***;

5) конечный угол ***alp1***;

6) шаг, с которым будут вычисляться значения ***x*** и ***y***.

Все вычисления производятся в цикле в 87–90 строках. Результаты заносятся в список ***vertex***, который и возвращается функцией ***circle*** в 91 строке.

```
85 def circle(x0, y0, r, alp0=0, alp1 = 360, resolution = 1, sector = 0):
86         vertex   []
87         for alp in range (alp0, alp1, resolution):
88                 x = x0 + r*math.cos(alp*math.pi/180)
89                 y = y0 + r*math.sin(alp*math.pi/180)
90                 vertex.append([x,y])
91         return vertex
```

*Рисунок XVI-1. Функция для выжигания окружности радиуса **r** с центром в координатах x_0, y_0*

Для выжигания окружности при использовании функции ***circle*** необходимо прежде всего получить координаты точек окружности, как это сделано в строке 113 на рисунке XVI-2, а затем осуществить и выжигание этих точек.

```
111 print ("START")
112
113 vertex = circle (222, 0, 100)
114 polygon (vertex)
115
116
117 print ("FINISH")
```

*Рисунок XVI-2. Вызов функции **circle** для расчета координат окружности и ее выжигание при помощи функции **polygon***

Результатом выполнения этого кода станет выжженная окружность (рисунок XVI-3).

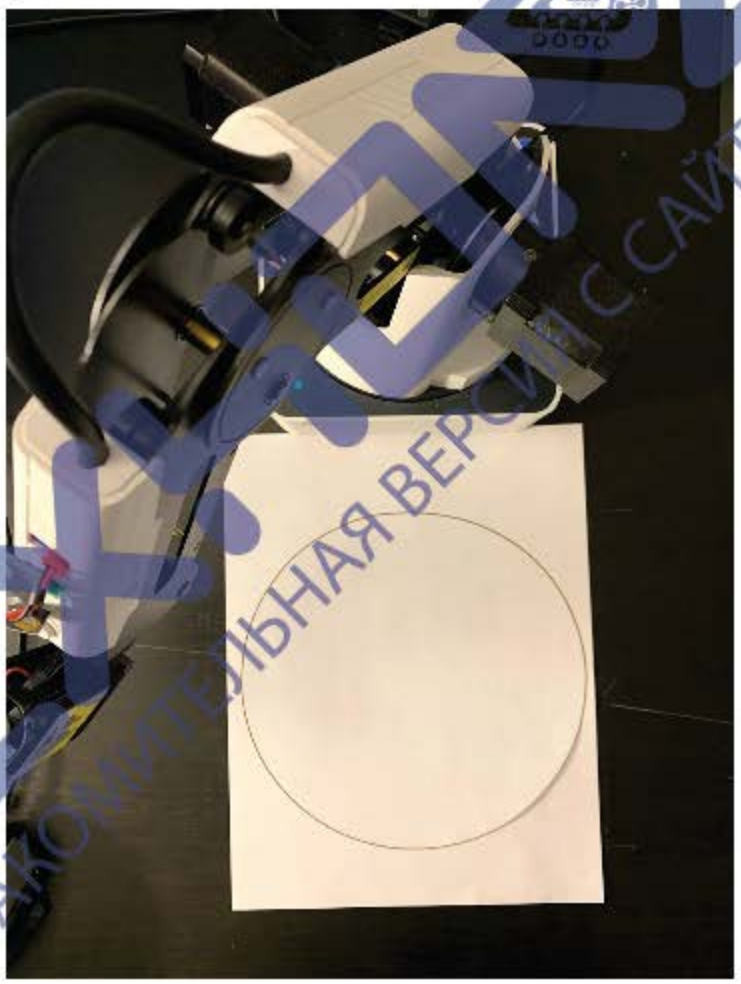


Рисунок XVI-3. Выжигание окружности при помощи Dobot

Однако функцию ***circle*** можно вызвать и с другими аргументами, например для того, чтобы выжечь не всю окружность, а лишь сегмент круга (рисунок XVI-4).

```
111 print ("START")
112
113 vertex = circle (222, 0, 100, 90, 180)
114 polygon (vertex)
115
116
117 print ("FINISH")
```

Рисунок XVI-4. Код для выжигания сегмента круга

В этом случае выжигание начнется с угла 90° относительно ***оси x***, а закончится на угле 180° (рисунок XVI-5).

Рисунок XVI-5. Сегмент круга, вырезанный при помощи Dobot

Можно ли использовать функцию ***polygon()*** для выжигания дуги, ведь эта функция рассчитана на то, чтобы в обязательном порядке осуществлять замыкание многогранника, то есть соединять первую и последнюю точки? Существует два способа решения этой проблемы. Первый — написать новую функцию и использовать ее. Второй — переделать саму функцию ***polygon()***, но сделать это необходимо таким образом, чтобы изменения в ней не привели к изменению работы других функций. Правильно использовать еще один аргумент, значение которого будет задано по умолчанию. В нашем случае это аргумент ***closed***, который по умолчанию равен 1. В случае если он не равен единице, замыкание многоугольника в строке 39 не происходит:

```
32 def polygon (vertex, closed = 1):
33             """Выжигает многоугольник с вершинами среди списков vertex"""
34             laser_line (0, vertex[0][0], vertex[0][1])
35             for v in vertex[1:]:
36                         laser_line (100, v[0], v[1], 3, 3)
37                         print (v[0], v[1])
38             if closed == 1:
39                         laser_line (100, vertex[0][0], vertex[0][1],3,3)
40
```

*Рисунок XVI-6. Модернизация функции **polygon()***

В таком случае код для выжигания будет выглядеть следующим образом:

```
117 print ("START")
118
119 vertex = circle (222, 0, 100, 270, 360,1,0)
120 polygon (vertex, 0)
121
122
123 print ("FINISH")
```

Рисунок XVI-7. Код для выжигания дуги

В 119 строке рассчитываются координаты дуги, а затем в 120 строке вызывается функция ***polygon()*** со вторым аргументом, равным 0. Таким образом, полученная линия не будет замкнутой.

Рисунок XVI-8. Дуга, выжженная при помощи Dobot

Осталось реализовать только сектор. Для этого необходимо начать выжигать из центра, а функция ***polygon()*** должна быть запущена в обычном режиме, то есть замыкать выжигаемую фигуру. Модернизация функции ***circle()*** показана на рисунке XVI-9. В случае если аргумент функции ***sector***, по умолчанию равный нулю, не равен нулю, первая точка для выжигания — центр окружности.

```
86 def circle(x0, y0, r, alp0=0, alp1 = 360, resolution = 1, sector = 0):
87             vertex   []
88             if sector != 0:
89                       vertex.append([x0,y0])
90             for alp in range (alp0, alp1, resolution):
91                       x = x0 + r*math.cos(alp*math.pi/180)
92                       y = y0 + r*math.sin(alp*math.pi/180)
93                       vertex.append([x,y])
94             return vertex
```

Рисунок XVI-9. Модернизация функции ***circle ()***

Код для выжигания сектора круга с охватом в 90° показан на рисунке XVI-10:

```
117 print ("START")
118
119 vertex   circle (222, 0, 100, 270, 360,1,0)
120 polygon (vertex, 0)
121
122
123 print ("FINISH")
```

Рисунок XVI-10. Код для выжигания сектора круга при помощи функций ***circle()*** *и* ***polygon()***

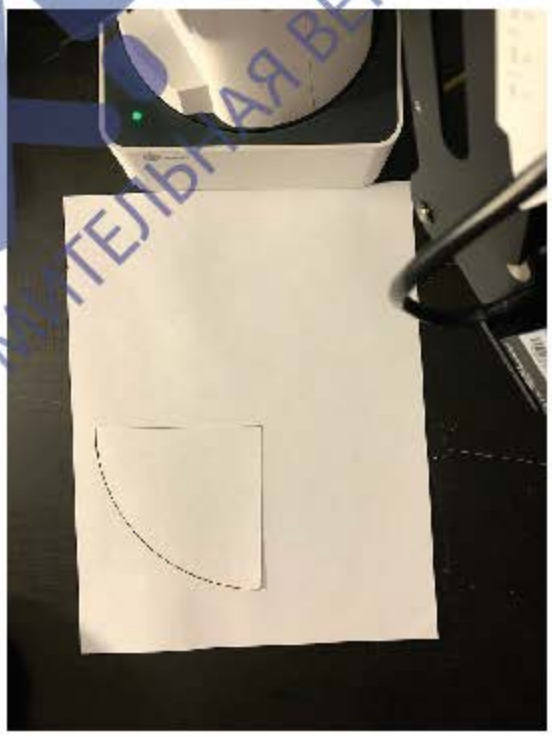


Рисунок XVI-11. Сектор круга, вырезанный при помощи Dobot

В следующем коде (рисунок XVI-12) в бесконечном цикле случайным образом выбирается угол, который будет ограничивать сектор, а также радиус этого сектора. В 14 строке осуществляется вырезание этого сектора. Начальный угол для выжигания задается в 118 строке — он равен 0, но каждый раз в цикле в 125 строке начальный угол увеличивается. Если сумма углов (начального и текущего случайного) больше 360°, то осуществляется выход из цикла (строки 122, 123). В 127 строке осуществляется дорисовывание до 360°.

```
117 print ("START")
118 alp0 = 0
119 while True:
120             alp = random.randint(0, 360)
121             r = random.randint (20,100)
122             if alp + alp0 > 360:
123                     break
124             vertex = circle (222, 0, r, alp0, alp + alp0, 1, 1)
125             alp0 = alp0 + alp
126             polygon (vertex)
127 vertex = circle (222, 0, r, alp0, 360, 1, 1)
128 polygon (vertex)
129
130 print ("FINISH")
```

Рисунок XVI-12. Код для выжигания случайных секторов

Рисунок XVI-13. Вырезание случайных секторов

Этот код можно немного модернизировать так, чтобы размер сектора в градусах был несколько меньше, например от 0° до 90°.

```
print ("START")
alp0    0
while True:
            alp = random.randint(0, 90)
            r = random.randint (20,100)
            if alp + alp0 > 360:
                        break
            vertex = circle (222, 0, r, alp0, alp + alp0, 1, 1)
            alp0 = alp0 + alp
            polygon (vertex)
vertex = circle (222, 0, r, alp0, 360, 1, 1)
polygon (vertex)

print ("FINISH")
```

В этом случае будет получен следующий рисунок.

Рисунок XVI-4. Вырезание случайных секторов

Задание для самостоятельной работы: при помощи функции ***circle()*** реализовать выжигание цифры 8 и латинской буквы S.

ЗАДАНИЕ XVII. Правильные многоугольники и окружность

Часто при создании графических редакторов отдельно выделяется класс многоугольников с одинаковыми сторонами — это так называемые правильные многоугольники. Давайте создадим функцию для определения вершин правильных многоугольников. Учтем, что в правильных многоугольниках равны между собой не только стороны, но и углы. Сумма углов правильных многоугольников равна 360°, а следовательно, несложно рассчитать и значение угла между сторонами.

```
 97 def equilateral_polygon (x, y, l, n):
 98             vertex = []
 99             vertex.append([x,y])
100             for alp in range (1,n):
101                         x = x + l * math.cos((alp*360*math.pi/180)/n)
102                         y   y + l * math.sin((alp*360*math.pi/180)/n)
103                         vertex.append([x,y])
104             return vertex
```

Рисунок XVII-1. Функция для построения правильных многоугольников

На рисунке XVII-1 показана функция, которая принимает значения координат начальной точки (***x***, ***y***), которая находится на одной из сторон, длины стороны (– ***l***), а также количество углов в многоугольнике (***n***). В 99 строке в список ***vertex*** заносятся координаты в первой строке, а затем в цикле рассчитывается положение каждой следующей вершины. При этом учитывается, что каждая следующая вершина находится на расстоянии ***l*** от предыдущей и образует с ней известный нам угол 360°/n. По сути, в строках 101 и 102 рассчитывается приращение по ***осям x*** и ***y***, а затем эти приращения прибавляются к координатам уже выжженной вершины. В строке 103 осуществляется добавление координат новой вершины в список **vertex**. Именно этот список и возвращает написанная функция.

```
124 print ("START")
128
129 vertex = equilateral_polygon (122,  100, 100, 3)
130 polygon (vertex)
131
132 print ("FINISH")
```

Рисунок XVII-2. Код для выжигания правильного многоугольника

На рисунке ниже показано, как осуществляется вызов функции ***equilateral_polygon()*** для начальной точки с координатами (122; – 100), длиной стороны 100 мм и количеством сторон 3. В результате получается фигура правильного многоугольника.

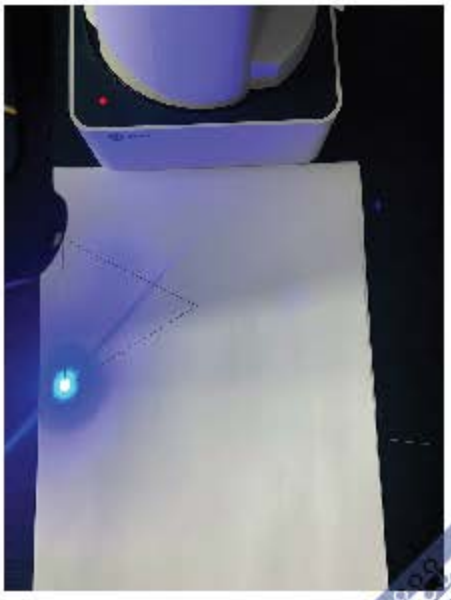


Рисунок XVII-3. Выжигание правильного многоугольника

Протестируем наш код для многоугольников с различным количеством сторон:

```
print ("START")

for i in [3,4,5,6,7,8]:
            vertex = equilateral_polygon (270,  100, 60, i)
            print (vertex)
            polygon (vertex)

print ("FINISH")
```

Рисунок XVII-4. Код для выжигания серии многоугольников

Согласно вышеописанному коду Dobot выжжет многоугольники с количеством сторон от 3 до 8 и длиной каждой стороны 60 мм.

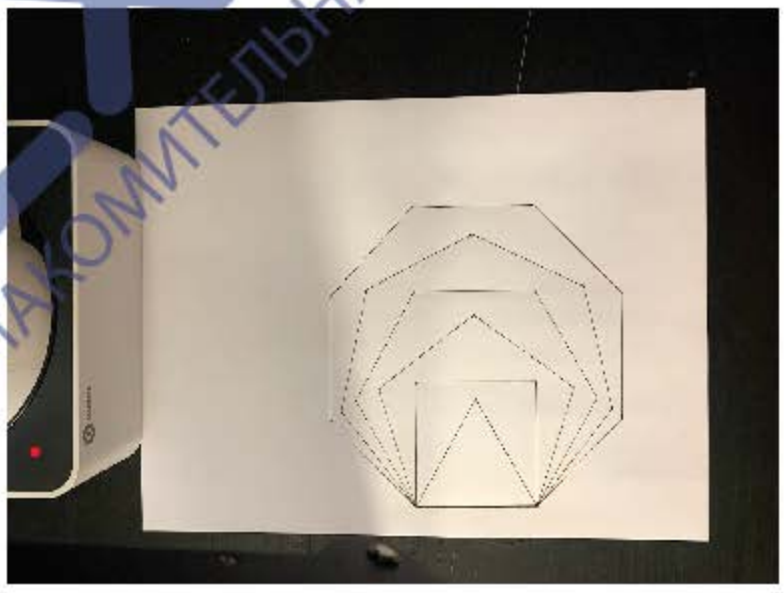


Рисунок XVII-5. Серия многоугольников с различным количеством сторон

Как мы можем видеть на рисунке XVII-5, наш код справляется с этой задачей, однако такая функция не всегда удобна. Для удобства необходимо предусмотреть возможность выжигать многоугольники из их середины. В данном случае середина — это центр описанной или вписанной окружности. А параметром, задающим размер многоугольника, удобнее всего сделать радиус этой окружности.

```
97 def equilateral_polygon (x, y, l, n, center = 0):
98         vertex = []
99         if center == 0:
100                 vertex.append([x,y])
101                 for alp in range (0,n):
102                         x = x + l * math.cos((alp*360*math.pi/180)/n)
103                         y = y + l * math.sin((alp*360*math.pi/180)/n)
104                         vertex.append([x,y])
105                 return vertex
106         else:
107                 xr = 0
108                 yr = 0
109                 for aln in range (0,n):
110                         xr = x + l*math.cos(n*2 * math.pi/n)
111                         yr = y + l*math.sin(n*2 * math.pi/n)
112                         vertex.append([xr,yr])
113                 return vertex
```

Рисунок XVII-6. Модернизация функции для расчета вершин правильных многоугольников

Модернизируем функцию для рисования правильных многоугольников. Введем аргумент ***center***, значение которого по умолчанию будет ***0***. В этом случае функция будет рассчитывать вершины правильных многоугольников тем же способом, что мы делали ранее. В любом другом случае расчет вершин многоугольника будет осуществляться исходя из радиуса окружности, в которую он вписан (строки 109–112).

```
136 print ("START")
137
138 for i in [3,4,5,6,7,8]:
139         vertex = equilateral_polygon (222, 0, 60, i,1)
140         print (vertex)
141         polygon (vertex)
142
143 print ("FINISH")
```

Рисунок XVII-7. Код для выжигания правильных многоугольников с общим центром

При выполнении кода для выжигания правильных многоугольников с общим центром получится серия многоугольников, такая как на рисунке XVII-8.

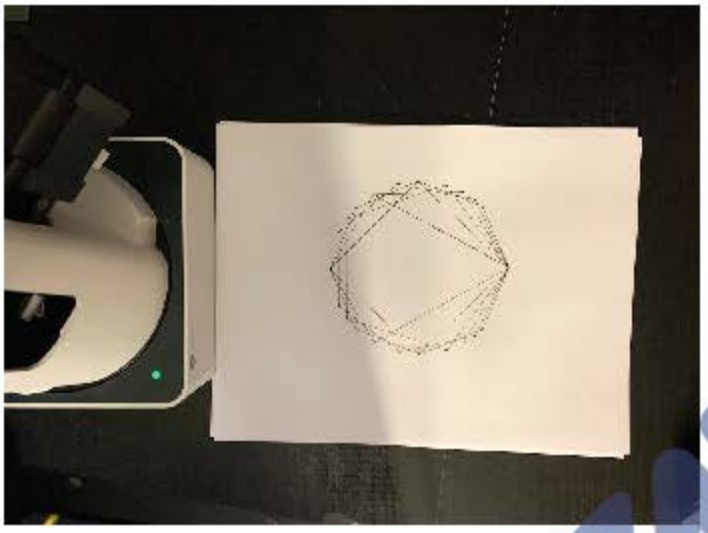


Рисунок XVII-8. Серия правильных многоугольников с общим центром

А какая фигура будет выжжена, если задать очень большое количество углов многоугольника, например 360?

```
136 print ("START")
137
138 vertex = equilateral_polygon (222, 0, 60, 360,1)
139 polygon (vertex)
140
141 print ("FINISH")
```

Рисунок XVII-9. Вызов функции для выжигания правильных многоугольников с числом сторон 360

В этом случае снова получится окружность, так как окружность можно рассматривать как правильный многоугольник с бесконечным количеством сторон.

Рисунок XVII-10. Окружность — правильный многоугольник с бесконечным количеством сторон

Задание для самостоятельной работы: модернизировать функцию для рисования правильных многоугольников так, чтобы была возможность задавать угол наклона первой стороны относительно осей координат.

ЗАДАНИЕ XVIII. Поворот и параллельный перенос геометрических фигур

В наших предыдущих заданиях в основном получались многоугольники, стороны которых параллельны одной из осей: ***x*** или ***y***. На этом занятии нам необходимо создать код, который позволит осуществлять разворот против часовой стрелки фигуры, вершины которой рассчитаны заранее. Например, проще всего рассчитать прямоугольник с двумя сторонами, параллельными ***оси x*** и ***оси y***, а затем осуществить разворот каждой из его вершин. Таким образом, все точки этого прямоугольника сначала будут рассчитаны в одной системе координат, а затем будет реализован разворот всей системы координат на определенный угол. Чтобы представить, как это происходит, обратимся к рисунку XVIII-1, демонстрирующему разворот оранжевой системы координат относительно синей. В оранжевой системе координат у точки координаты — ***x'***, ***y'***. Чему будут равны координаты этой точки в синей системе координат?

Обратите внимание: красный и синий треугольники подобны, так как их стороны взаимно перпендикулярны. Координата ***x*** — это разность между областями, охватываемыми красной и оранжевыми фигурными скобками. Область красной фигурной скобки из синего прямоугольного треугольника равна:

$$y' \cdot sin(\omega),$$

тогда как оранжевая область равна:

$$x' \cdot cos(\omega).$$

Следовательно, координата ***x*** преобразится к виду:

$$x = x' \cdot cos(\omega) - y' \cdot sin(\omega).$$

Что касается координаты ***y***, то ее значение — это сумма областей, выделенных зеленой и синей фигурными скобками.

Синяя область — это

$$y' \cdot cos(\omega);$$

зеленая область —

$$x' \cdot sin(\omega).$$

Для координаты ***y*** получаем:

$$y = y' \cdot cos(\omega) + x' \cdot sin(\omega).$$

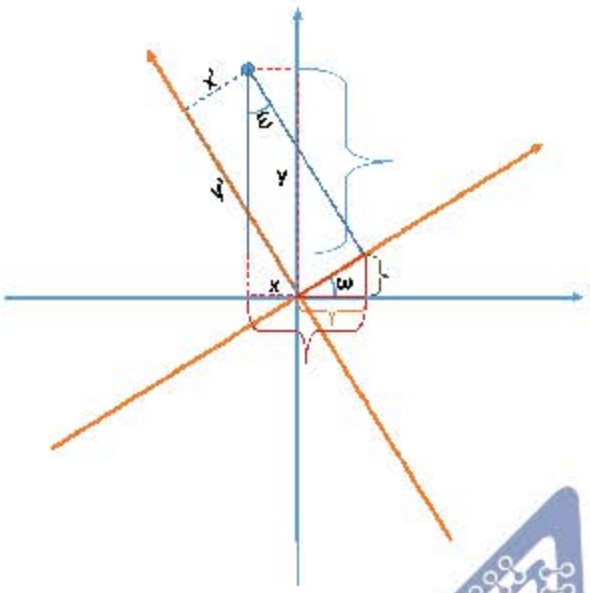


Рисунок XVIII-1. Разворот оранжевой системы координат относительно синей

Таким образом, для того чтобы осуществить поворот какой-либо фигуры, необходимо преобразовать каждую из ее вершин по следующему правилу:

$$x = x' * cos(\omega) - y' * sin(\omega),$$
$$y = y' * cos(\omega) + x' * sin(\omega),$$

где ω — это угол поворота против часовой стрелки. Обратите внимание, что данный подход работает только при повороте **против** часовой стрелки.

```
106 def turn (vertex, alp):
107             alp = alp*math.pi/180
108             turn_vertex = [[i[0]*math.cos(alp) - i[1]*math.sin(alp),
109                             i[1]*math.cos(alp) + i[0]*math.sin(alp)] for i in vertex]
110             return turn_vertex
```

Рисунок XVIII-2. Код для поворота системы координат против часовой стрелки

На рисунке XVIII-2 представлена функция ***turn()***, которая получает в качестве аргумента координаты точек и угол, на который необходимо произвести разворот. В 107 строке этот угол переводится из градусов в радианы. Затем в 108 и 109 строках генерируется новый список. Для каждой из точек высчитываются новые координаты по формулам, описанным выше. Функция возвращает список ***turn_vertex*** с новыми координатами.

Удобнее всего этой функцией пользоваться следующим образом. Сначала сформировать список с координатами вершин. Эти координаты должны быть около ***0***. Затем осуществить разворот: разворот произойдет вокруг координат ***0; 0***. После этого осуществить перемещение полученных точек. Для этого необходимо к каждой координате ***x*** прибавить одно и то же число. То же касается и координаты ***y***. Это можно делать каждый раз «вручную», но лучше будет использовать специальную функцию ***move()***: например для Dobot центр координат — это ***(222,0)***. Реализацию этой функции для параллельного переноса вершин ***vertex*** по осям координат можно увидеть на рисунке XVIII-3:

```
111 def move (vertex, x, y):
112             move_vertex   [[i[0] + x, i[1] + y] for i in vertex]
113             return move_vertex
```

*Рисунок XVIII-3. Функция **move()** для параллельного переноса вершин **vertex** по осям координат*

В этой функции также генерируется список. Каждая координата, полученная из аргумента ***vertex***, преобразуется таким образом, что к ней добавляется определенное число ***x*** или ***y***, которое является одним из аргументов функции ***move ()***.

Рассмотрим пример совместного использования функций ***turn()*** и ***move()***.

```
136 print ("START")
137
130 for i in range (0,280,90):
139             vertex = equilateral_polygon (0, 0, 100, 3)
140             vertex = turn(vertex, i)
141             vertex = move(vertex,222,0)
142             polygon (vertex)
143
144 print ("FINISH")
```

*Рисунок XVIII-4. Совместное использование функций **turn()** и **move()***

При их совместном использовании цикл из строки 138 будет выполнен 4 раза. А переменная ***i*** примет значения ***0***, ***90***, ***180***, ***270***. Для каждого из этих значений в 139 строке будут рассчитаны вершины правильного треугольника, затем в 140 строке эти вершины будут развернуты на необходимый угол, в 141 строке произойдет перенос полученных вершин на 222 единицы по ***оси x***. Иными словами, сначала осуществляется расчет вершин треугольника около координат ***0***; ***0***. Затем осуществляется поворот этого треугольника и его перемещение к координате ***0***; ***222***. В результате получится четыре правильных прямоугольника, развернутых относительно друг друга на 90°.

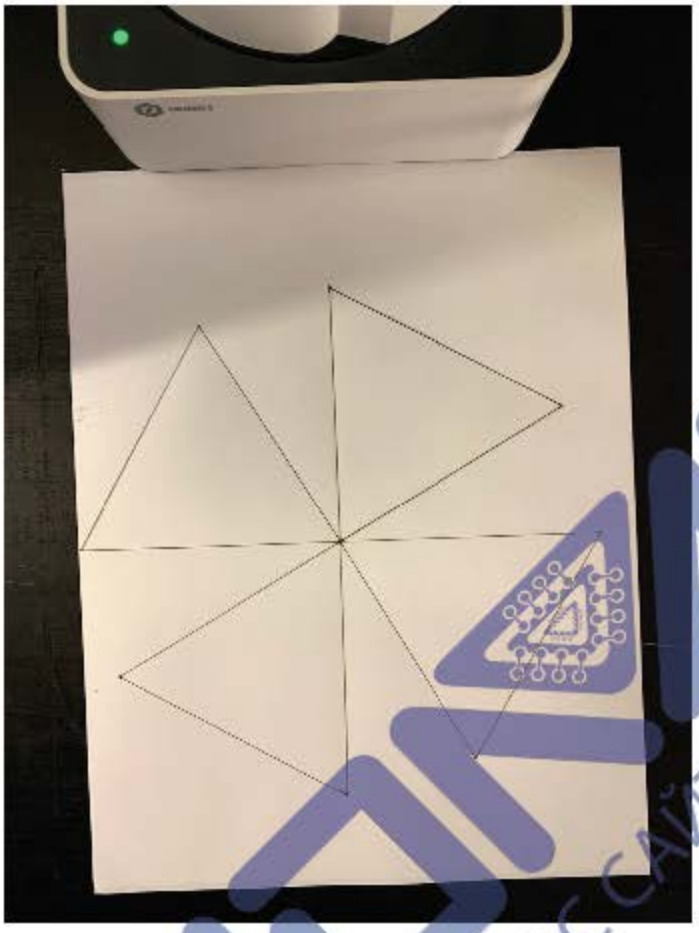


Рисунок XVIII-5. Четыре правильных прямоугольника, развернутых относительно друг друга на 90°

Обратите внимание: разворот происходит относительно одной из вершин треугольника — эта вершина находится в центре выжженного рисунка. В ней все треугольники касаются друг друга.

Для выжигания серии квадратов с общим центром, развернутых относительно друг друга, в коде разработан иной подход.

```
137 print ("START")
138
139 for i in range (0, 100, 10):
140             vertex = [[ 50, 50],[50, 50],[50,50],[ 50,50]]
141             vertex = turn (vertex, i)
142             vertex = move(vertex,222,0)
143             polygon (vertex)
144
145 print ("FINISH")
```

Рисунок XVIII-6. Серия квадратов с общим центром, развернутых друг относительно друга

В 140 строке каждый раз при выполнении цикла задаются координаты вершин квадрата, у которого пересечение диагоналей находится в точке ***0***, ***0***. Затем осуществляется разворот этого квадрата на угол ***i*** и перенос его в точку ***0***; ***222***. В строке 143 осуществляется выжигание текущего квадрата.

Получается, что соседние квадраты смещены относительно друг друга на 10°. В результате выполнения данного кода у нас получатся вписанная и описанная вокруг этих квадратов окружности (рисунок XVIII-7).

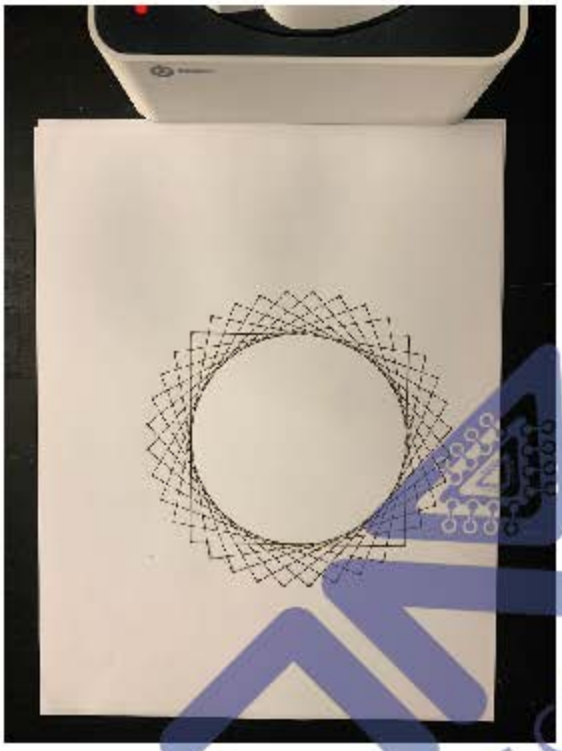


Рисунок XVIII-7. Серия квадратов, развернутых относительно друг друга

Задание для самостоятельной работы: осуществить выжигание каких-либо орнаментов, которые будут получены в ходе параллельного переноса или поворота фигур.

ЗАДАНИЕ XIX. Имитация библиотеки turtle для выжигания

В языке программирования Python заложена библиотека для рисования ***turtle()***. Она называется так потому, что для рисования в ней используется пиктограмма черепашки. В этой библиотеке реализованы несколько базовых функций, таких как:

- ✓ поворот на угол относительно предыдущего перемещения;
- ✓ движение пера вперед относительно предыдущего перемещения и заданного угла;
- ✓ поднять перо;
- ✓ опустить перо;
- ✓ скорость перемещения пера.

В сети Интернет и видеохостинге YouTube есть огромное количество заданий для этой библиотеки: от рисования простых линий до создания фракталов. Давайте попытаемся создать библиотеку, подобную ***turtle()***, для выжигания при помощи Dobot.

Для выполнения нам понадобится осуществить движение рабочего инструмента между двумя точками таким образом, чтобы учитывать его предыдущее перемещение. Иными словами, создать функцию, у которой будет четыре аргумента: это координаты начальной точки для перемещения (***x; y***), длина линии ***r***, которую необходимо выжечь, а также угол наклона выжженной линии относительно ***оси x*** — ***alp***. Эта функция должна возвращать текущее значение для положения рабочего инструмента.

```
115 def turtle(x, y, r, alp):
116         alp = alp*math.pi/180
117         laser_line (0, x, y, V = 20, a = 20)
118         x,y = x+r*math.cos(alp), y+r*math.sin(alp)
119         laser_line (100, x, y, V = 3, a = 3)
120         return(x,y)
```

*Рисунок XIX-1. Функция **turtle()** для Dobot*

В 116 строке осуществляется перевод привычных градусов в радианы. Затем в 117 строке происходит перемещение рабочего инструмента в положение, заданное первым и вторым аргументами функции ***turtle()*** с выключенным лазером, в 118 строке вычисляются координаты для места перемещения исходя из текущих координат — ***x***, ***y***, длины перемещения ***r*** и угла ***alp***. В 119 строке осуществляется выжигание. Функция возвращает координату рабочего инструмента в 120 строке. Применим эту функцию для выжигания пятиконечной звезды:

```
140 print ("START")
141
142 x,y = 150, -50
143 for i in range(5):
144             x,y = turtle(x, y, 50 ,i*144)
145
146 print ("FINISH")
```

Рисунок XIX-2. Код для выжигания пятиконечной звезды

Dobot выжжет 5 линий с углом 144 градуса между ними, все линии будут одинаковой длины — 50 мм (рисунок XIX-3):

Рисунок XIX-3. Пятиконечная звезда, выжженная при помощи Dobot

Настало время составить код для выжигания звезды с любым количеством лучей, например с нечетным, где количество лучей ***n*** = 7. В 146 строке вычисляется угол между прямыми образующими лучами. Это целое от деления количества лучей на 2, умноженное на 360 градусов и снова на количество углов.

```
143 print ("START")
144
145 n = 7
146 alp = n // 2 * 360 / n
147
148 x,y    150, -65
149 for i in range(n):
150             x,y   turtle(x, y, 150,i*alp + (180-alp) //2)
151
152 print ("FINISH")
```

Рисунок XIX-4. Код для выжигания звезды с нечетным количеством лучей

В цикле рисование начинается таким образом, чтобы первая прямая первого луча была под углом в половину от угла между лучами. В результате выполнения данного кода Dobot выжжет семиконечную звезду.

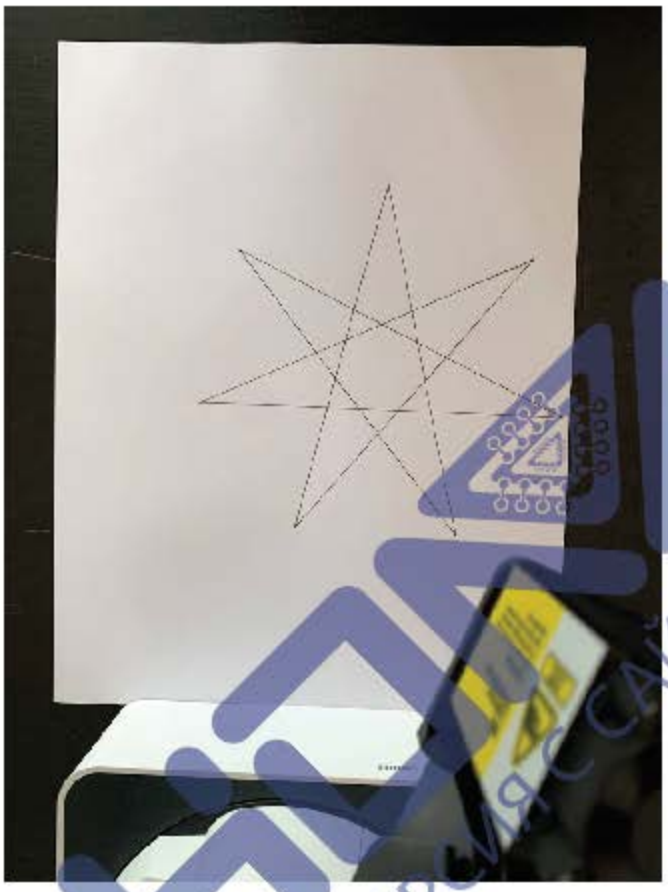


Рисунок XIX-5. Семиконечная звезда, выжженная при помощи Dobot

Примеры со звездами или другими фигурами очень распространены для библиотеки ***turtle*** языка Python, поэтому у вас есть возможность выполнить все эти задания самостоятельно. Однако полученная нами функция получилась не в формате подходов, рассмотренных в данном пособии. Изначально нами были предприняты попытки выполнить последовательность действий:

— вычисление координат вершин;

— разворот или перемещение полученных координат;

— выжигание фигуры.

В этом случае нам необходимо переделать функцию ***turtle()*** таким образом, чтобы она не выжигала, а только возвращала новые координаты. Кроме того, необходимо создать функцию ***star()***, которая будет использовать функцию ***turtle()*** для расчета вершин.

```
def turtle(x, y, r, alp):
        alp = alp/math.pi/180
        x,y = x+r*math.cos(alp), y+r*math.sin(alp)
        return(x,y)

def star (r, n, x, y):
        alp = n // 2 * 360 / n
        vertex = [[x,y]]
        print (vertex [0][0])
        for i in range(n-1):
                vertex.append([turtle(vertex[ i][0], vertex[ i][1], r,i*alp + (180-alp) //2)[0],
                               turtle(vertex[ i][0], vertex[ i][1], r,i*alp + (180-alp) //2)[1]])
        return (vertex)
```

*Рисунок XIX-6. Функции **turtle()** и **star()***

Теперь, как мы можем убедиться, функция ***turtle()*** получает и возвращает пару координат (***x; y***). А функция ***star*** вычисляет необходимые углы для выжигания звезды с заданным количеством лучей. При этом начальные координаты первого луча — это аргументы функции: ***x***, ***y***. Затем в цикле при помощи функции ***turtle()*** вычисляются координаты последующих вершин. Для этого в качестве аргументов ***x***, ***y*** используются координаты предыдущих вершин:

для ***x*** это ***vertex [–1][0],***
для ***y*** — ***vertex [–1][0].***

Проиллюстрируем преимущества такого подхода.

```
print ("START")

for x  in range (140, 311, 40):
            for y  in range (-90, 100, 40):
                        vertex = star (20,5, 0, 0)
                        vertex = turn(vertex, random.randint(0,360))
                        vertex = move (vertex, x, y)
                        polygon (vertex)


print ("FINISH")
```

Рисунок XIX-7. Код для выжигания множества пятиконечных звезд, развернутых случайным образом относительно друг друга

На рисунке XIX-7 представлен код, где перебираются координаты ***x*** и ***y*** с шагом в 40. Каждый раз для очередных пар координат рассчитывается положение звезды, затем ее координаты поворачиваются на случайный угол, после чего координаты звезды изменяются функцией ***move()***.

В результате получается скопление пятиконечных звезд, случайным образом развернутых по отношению друг к другу (рисунок XIX-8). Обратите внимание: на первый взгляд создается впечатление, что звезды расположены в случайном порядке. Однако такое их положение строго определено: разворот координат вершин каждой из звезд происходит относительно ее первой вершины.

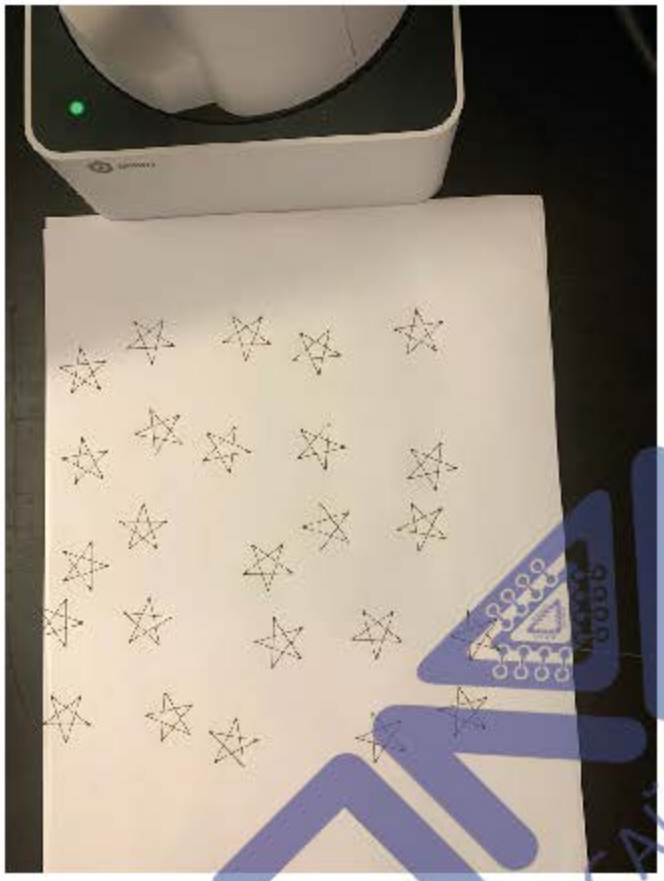


Рисунок XIX-8. Пятиконечные звезды, случайным образом развернутые относительно друг друга

Задание для самостоятельной работы: создать функцию, которая выжигает звезду с четным количеством лучей; создать так называемую снежинку Коха.

ЗАДАНИЕ XX. Библиотека primitives

Это последнее задание, и в нем нам необходимо из всех написанных функций собрать библиотеку. Она будет называться **primitives**(). Код этой библиотеки представлен ниже:

```
import threading
import DobotDllType as dType
import math
import random

def laser_point (power, time):
        """Выжигает определенное в текущем положении рабочего инструмента"""
        dType.SetEndEffectorLaser(api, 1, power, 1)
        dType.dSleep(time)
        dType.SetEndEffectorLaser(api, 0, power, 1)

def laser_line (power, x, y, V = 20, a = 20):
        """Выжигает линию или перемещается из текущей координаты в координаты (x,y)"""
        dType.SetPTPCommonParams(api, v, a, 1)
        if power    0:
                a = 0
        else:
                a = 1
        dType.SetEndEffectorLaser(api, a, power, 1)
        dType.SetPTPCmd(api, 2, x, y, 0, 0, 1)
        dType.SetEndEffectorLaser(api, 0, 0, 1)

def rectangle (x, y, length, width, V = 5, a = 5):
        """Выжигает прямоугольник с центром в x, y"""
        x   x + length / 2
        y   y - width / 2
        laser_line (0, x, y)
        laser_line (100, x   length, y, V, a)
        laser_line (100, x - length, y - width, V, a)
        laser_line (100, x, y - width, V, a)
        laser_line (100, x, y, V, a)

def polygon (vertex, closed = 1):
        """Выжигает многоугольник с вершинами, заданы списком vertex"""
        laser_line (0, vertex[0][0], vertex[0][1])
        for v in vertex[1:]:
                laser_line (100, v[0], v[1], 5, 5)
                print (v[0], v[1])
        if closed == 1:
                laser_line (100, vertex[0][0], vertex[0][1],5,5)

def numbers (n):
        """Принимает на вход цифру и выжигает ее лазером в прямоугольнике 4x6"""
        n0 = [1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0]

        n1 = [0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1]

        n2 = [0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1]

        n3 = [0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0]

        n4 = [0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1]

        n5 = [1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0]

        n6 = [0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0]

        n7 = [1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1]

        n8 = [0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0]

        n9 = [0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0]

        num = [n0, n1, n2, n3, n4, n5, n6, n7, n8, n9]

        return num[n]
```

```
def numbers_laser (num, x,y, size):
        """Выкладем цифру размером size в точке x,y"""
        i=0
        print (num)
        n = numbers (num)

        for yt in range (y, y + size *4, size):
                if n [i] ==1:
                        rectangle (xt, yt, size, size)
                i+=1

def triangle (a, alpha, betta):
        """Находим координаты трех вершин треугольника,
        при этом первая вершина находится в точке (0,0), вторая - на оси x"""
        c = a * math.sin(betta * math.pi / 180) / math.sin ((alpha + betta) * math.pi / 180)
        x2 = c * math.cos(alpha * math.pi / 180)
        y2 = c * math.sin(alpha * math.pi / 180)
        return [[0,0],[a,0],[x2,y2]]

def circle(x0, y0, r, alp0=0, alp1 = 360, resolution = 1, sector = 0):
        """Возвращает координаты для вывода окружности, сектора, сегмента, дуги"""
        vertex = []
        if sector != 0:
                vertex.append([x0,y0])
        for alp in range (alp0, alp1, resolution):
                x = x0 + r*math.cos(alp*math.pi/180)
                y = y0 + r*math.sin(alp*math.pi/180)
                vertex.append([x,y])
        return vertex

def equilateral_polygon (x, y, l, n):
        """Возвращает координаты правильного многоугольника"""
        vertex = []
        vertex.append([x,y])
        for alp in range (1,n):
                x = x + l * math.cos((alp*360*math.pi/180)/n)
                y = y + l * math.sin((alp*360*math.pi/180)/n)
                vertex.append([x,y])
        return vertex

def turn (vertex, alp):
        """Осуществляет разворот многоугольника на угол alp"""
        alp = alp*math.pi/180
        turn_vertex = [[i[0]*math.cos(alp) - i[1]*math.sin(alp),
                        i[0]*math.sin(alp) + i[1]*math.cos(alp)] for i in vertex]
        return turn_vertex

def move (vertex, x, y):
        """Осуществляет перенос многоугольника, заданного вершинами vertex, на x и y по ОХ"""
        move_vertex = [[i[0] + x, i[1] + y] for i in vertex]
        return move_vertex

def turtle(x, y, l, alp):
        """ Возвращает точку, получающуюся при перемещении т. от точки0, с учетом угла alp"""
        alp = alp*math.pi/180
        x,y = x+l*math.cos(alp), y+l*math.sin(alp)
        return (x,y)

def star (n, m, x, y):
        """Возвращает координаты вершин n-конечной звезды из начальной точки x, y"""
        alp = m // 2 * 360 / n
        vertex = [x,y]
        print (vertex [0][0])
        for i in range(n-1):
                vertex.append([turtle(vertex[-1][0], vertex[-1][1], i*alp + 180 alp) //2)[0],
                               turtle(vertex[-1][0], vertex[-1][1], i*alp + 180 alp) //2)[1]])
        return (vertex)


CON_STR = {
    dType.DobotConnect.DobotConnect_NoError:  "DobotConnect_NoError",
    dType.DobotConnect.DobotConnect_NotFound: "DobotConnect_NotFound",
    dType.DobotConnect.DobotConnect_Occupied: "DobotConnect_Occupied"}
#Load Dll
api = dType.load()
#Connect Dobot
state = dType.ConnectDobot(api, "", 115200)[0]
print("Connect status:",CON_STR[state])
if (state == dType.DobotConnect.DobotConnect_NoError):
        #Clean Command Queued
        dType.SetQueuedCmdClear(api)
        #Async Motion Params Setting
        dType.SetHOMEParams(api, 200, 0, 0, 0, isQueued = 0)
        dType.SetHOMECmd(api, 0, isQueued=1)
        dType.SetPTPCommonParamsEx(api,20,20,1)
```

Файл с данной программой необходимо сохранить в корневую папку языка Python. Мы назвали этот файл по названию библиотеки — ***primitives***.

В таком случае файл будет рассматриваться интерпретатором Python как библиотека, следовательно, мы сможем осуществлять работу со всеми функциями как с библиотекой.

Давайте рассмотрим пример, в котором необходимо выжечь так называемую розу ветров — четырехлучевую звезду.

```
1 import primitives as pr
2 wind_rose =[]
3 for i in range(4):
4             vertex = [[0,0],[10,10],[0,30],[-10,10]]
5             vertex = pr.turn(vertex,i*90)
6             vertex = pr.move(vertex, 222, 0)
7             wind_rose.extend(vertex)
8 pr.polygon(wind_rose)
```

Рисунок XX-1. Код для выжигания розы ветров

В первой строке вызывается библиотека ***primitives***. Ее имя можно использовать в сокращенном виде — ***pr***.

Затем создается массив ***wind_rose***. Каждый раз в цикле создается луч звезды. При этом начальная точка — 0,0. Затем луч разворачивается кратно 90°, после чего происходит его перемещение в точку 222; 0. В 7 строке координаты вершин очередного луча добавляются к координатам уже рассчитанных вершин лучей. В 8 строке осуществляется выжигание полученной фигуры.

Рисунок XX-2. Роза ветров

Итак, нам осталось только собрать воедино все возможности, которые открывает перед нами написанная библиотека:

1) выжигание точки;
2) перемещение рабочего инструмента при выключенном лазере;
3) перемещение рабочего инструмента с включенным лазером (выжигание);
4) выжигание цифр;
5) расчет вершин треугольников;
6) расчет координат для выжигания окружности, сегмента, дуги и сектора;
7) расчет вершин правильных многоугольников;
8) разворот фигур против часовой стрелки;
9) перемещение фигур по осям;
10) имитация библиотеки ***turtle***.

Таким образом, наша библиотека позволяет создавать любые плоские геометрические фигуры, а также их сочетания.

Задание для самостоятельной работы: модернизировать библиотеку ***primitives*** для создания новых объектов, например букв, спиралей, лабиринтов, составных фигур.