Learn Python Programming

This site contains materials and exercises for the Python 3 programming language.
In this course you will learn how to write code, the basics and see examples.

Python is a programming language supports several programming paradigms including Object-Orientated Programming (OOP) and functional programming.

Related course: Complete Python Programming Course & Exercises

Table of Contents:

Overview of articles and exercises:

Variables and Types

Python supports *different types of variables* (datatypes) such as whole numbers, floating point numbers and text.

You do not need to specify the datatype of a variable, you can simply assign any value to a variable. Type the program below and start it.

Related course: Complete Python Programming Course & Exercises

Datatypes

Variables can be of several data types. Python supports integers (numbers), floating point numbers, booleans (true or false) and strings (text).

Python will determine the datatype based on the value you assign to the variable. If you create a variable x, x = 3, then Python assumes its an integer. But if you assign x = 1.5 then Python knows its not an integer but floating point number.

Example

The example below shows you several variables. These can be assigned as you wish. Once defined you can print them or use arithmetics.

```
#!/usr/bin/python

x = 3                 # a whole number
f = 3.1415926         # a floating point number
name = "Python"       # a string

print(x)
print(f)
print(name)

combination = name + " " + name
print(combination)

sum = f + f
print(sum)
```

Run the program from terminal or with an IDE.

```
python example.py
```

In the example we have several variables (x,f,name) which are of different data types. Later in the program we create more variables (combination, sum).

Variables can be defined anywhere in the program. Variable names can be one to n letters.

Python Strings (With Examples)

Any time you want to use *text* in Python, you are using *strings*. Python understands you want to use a string if you use the double-quotes symbol.

Once a string is created, you can simply print the string variable directly. You can access characters using block quotes.

Related course: Complete Python Programming Course & Exercises

Strings

Define string

Variables can be of the string data type. They can hold characters or text.
If you create string variable x. You can show it on the screen using the print() function.

```
x = "Hello"
print(x)
```

String indexing

Individual characters can be accessed using blockquotes, counting starts from zero.

```
print(x[0])
print(x[1])
```

The first character starts at zero. This may be a bit counter intuitive, but has historic reasons.

Sub string

By using a colon you can create a substring. If no start or end number is written, Python assumes you mean the first character or last character.

Try the example below:

```
x = "hello world"
s = x[0:3]
print(s)
s = x[:3]
print(s)
```

Complete example

This example does a lot of string operations like printing text, numbers, combining strings, slicing and accessing elements.

Try the program below:

How to read keyboard-input?

In Python and many other programming languages you can get user input. Do not worry, you do not need to write a keyboard driver.

The **input()** function will ask keyboard input from the user. If you are still using Python 2, you have the function raw_input().

Related course: Complete Python Programming Course & Exercises

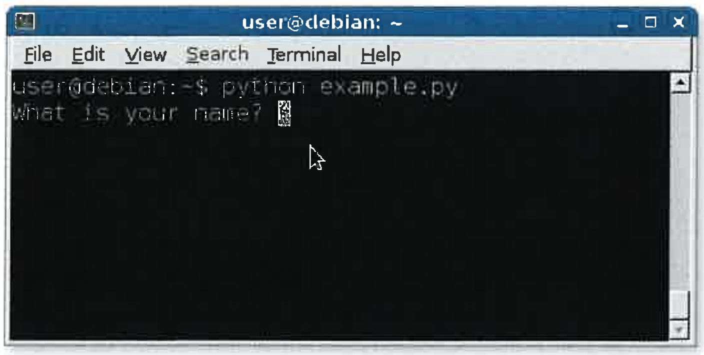
Example

The input function prompts text if a parameter is given. The functions reads input from the keyboard, converts it to a string and removes the newline (Enter).

Type and experiment with the script below (save as key.py)

```
#!/usr/bin/env python3

name = input('What is your name? ')
print('Hello ' + name)

job = input('What is your job? ')
print('Your job is ' + job)

num = input('Give me a number? ')
print('You said: ' + str(num))
```

Output should be something like this, depending on your terminal:



```
user@debian:~$ python example.py
What is your name?
```

By the time you are reading this, perhaps you are used to voice input or other types of human-computer interaction. Eitherway keyboard input is still very useful for coding.

If you are a beginner, then I highly recommend this book.

Exercise

Try these exercises:

1. Make a program that asks a phone number.

If Statements Explained

A program sometimes may have to make choices. These choices can execute different code depending on certain condition.

In Python the **if statement** is used for conditional execution or branching. An if statement is one of the **control structures**. (*A control structure controls the flow of the program.*)

The if statement may be combined with certain operator such as equality (==), greater than (>=), smaller than (<=) and not equal (!=). Conditions may be combined using the keywords **or** and **and**.

Related course: Complete Python Programming Course & Exercises

Introduction

In the example below we show the use *if* statement, a control structure. An if statement evaluates data (a condition) and makes a choice.

Lets have al look at a basic if statement. In its basic form it looks like this:

```
#!/usr/bin/env python3
if <condition>:
    <statement>
```

In this form

- is the condition evaluated as a Boolean, it can either be True or False.
- is one more lines of code. Each of those lines must indented with four spaces.

Several examples of the if statements are shown below, you can run them in the Python interpreter:

```
#!/usr/bin/env python3
>>> x = 3
>>> if x < 10:
...     print('x below ten')
...
x below ten
>>> if x > 10:
...     print('x is greater than ten')
...
>>> if x > 1 and x < 4:
...     print('x is in range')
...
x is in range
>>>
```

It's very important to have four spaces for the statements. Every if statement needs a colon. More than one condition can be combined using the *and* keyword.

Indentation and Blocks

An if statement doesn't need to have a single statement, it can have a **block**. A block is more than one statement.

The example below shows a code block with 3 statements (print). A block is seen by Python as a single entity, that means that if the condition is true, the whole block is executed (every statement).

```
#!/usr/bin/env python3
x = 4
if x < 5:
    print("x is smaller than five")
    print("this means it's not equal to five either")
    print("x is an integer")
```

All programming languages can create blocks, but Python has a unique way of doing it. A block is defined only by its indention.

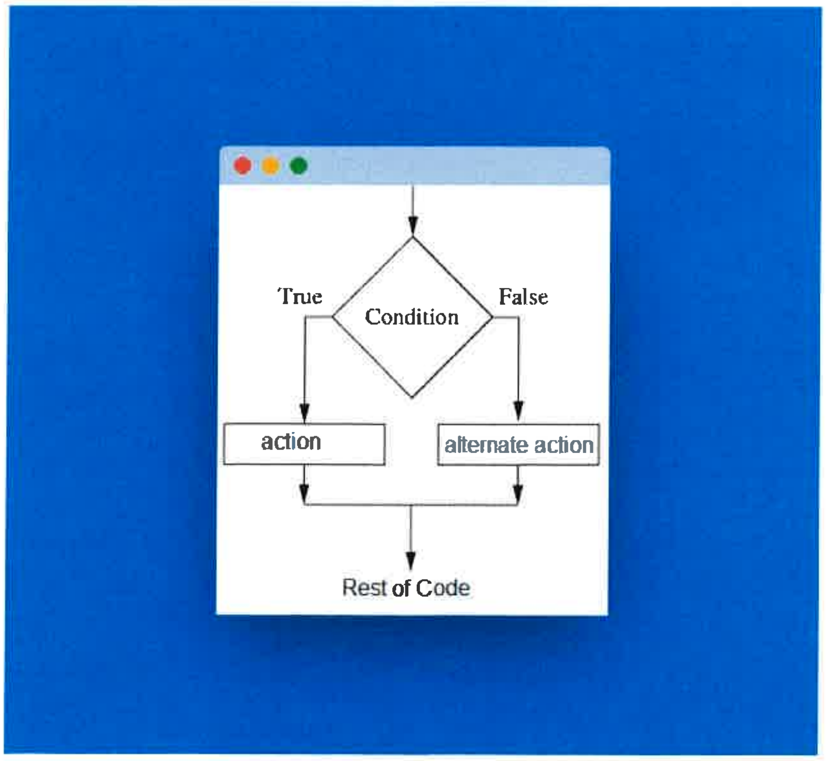
Other programming languages often used symbols like `{`, `}` or words `begin` and `end`.

So the basic form of a Python if statement block is:

```
if <condition>:
    <statement>
    <statement>
    <statement>

<statement>  # not in block
```

After completing the if statement, Python continues execution of the program. The if statement ends by its indetion, it goes back four spaces.

Visual example of if statement (click to enlarge):



Python "for" Loops (Iteration Introduction)

Programs sometimes need to repeat actions. To repeat actions we can use a **for loop**.
A for loop is written inside the code. A for loop can have 1 or more instructions.

A for loop will repeat a code block. Repeation is continued until the stop condition is met. If the stop condition is not met it will loop infintely.

These instructions (loop) is repeated until a condition is met.

Related course: Complete Python Programming Course & Exercises

Example

In the exercise below we will repeat actions on every item of a list.

The first loop will repeat the print functionfor every item of the list.
The second loop will do a calculation on every element of the list num and print the result.

Type the code below and run the program.

```
#!/usr/bin/env python3

city = ['Tokyo','New York','Toronto','Hong Kong']
print('Cities loop:')
for x in city:
    print('City: ' + x)

print('\n')  # newline

num = [1,2,3,4,5,6,7,8,9]
print('x^2 loop:')
for x in num:
    y = x * x
    print(str(x) + '*' + str(x) + '=' + str(y))
```

Save the file as loopexample.py
Then run the code with the command:

```
python loopexample.py
```

Schematically a for loop does this:

Python "while" Loops (Indefinite Iteration)

A while loop repeats code until the condition is met. Unlike for loops, the number of iterations in it may be unknown. A while loop always consists of a condition and a block of code.

A while loop ends if and only if the condition is true, in contrast to a for loop that always has a finite countable number of steps.

Related course: Complete Python Programming Course & Exercises

Example

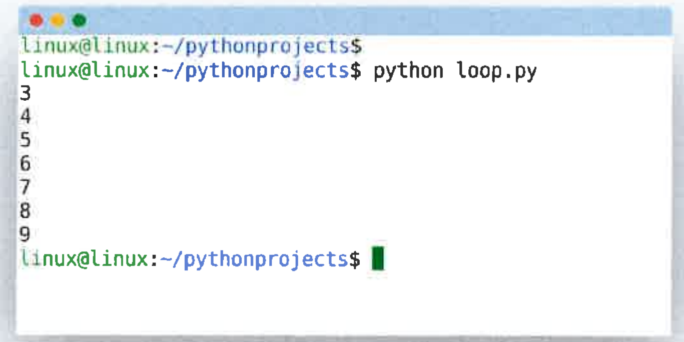
While loop example

The while loop below defines the condition (x < 10) and repeats the instructions until that condition is true. Type this code:

```
#!/usr/bin/python

x = 3
while x < 10:
    print(x)
    x = x + 1
```

Executes the code below until the condition x < 10 is met. Unlike a for loop, the iterator i is increased in the loop.

Save then run with your Python IDE or from the terminal.



```
linux@linux:~/pythonprojects$
linux@linux:~/pythonprojects$ python loop.py
3
4
5
6
7
8
9
linux@linux:~/pythonprojects$
```

You can also create infinite loops, this is when the condition never changes.

Functions in Python (With Examples)

To group sets of code you can use functions. Functions are small parts of repeatable code. A function accepts parameters.

Without functions we only have a long list of instructions. Functions can help you organize code. Functions can also be reused, often they are included in modules.

Related course: Complete Python Programming Course & Exercises

Example

Functions

Functions can be seen as executable code blocks. A function can be used once or more.

A simple example of a function is:

```
def currentYear():
    print('2018')

currentYear()
```

The function is immediately called in this example. Function definitions always start with the def keyword.

Functions can be reusable, once created a function can be used in multiple programs. The print function is an example of that.

Functions with parameters

In the example below we have parameter x and y. Type this program and save it as summation.py

```
#!/usr/bin/env python3

def f(x,y):
    return x*y

print(f(3,4))
```

In this example we have two functions: f(x,y) and print(). The function f(x,y) passed its output to the print function using the *return* keyword.

Python Lists (With Examples)

List can be seen as a collection: they can hold many variables. List resemble physical lists, they can contain a number of items.

A list can have any number of elements. They are similar to arrays in other programming languages. Lists can hold all kinds of variables: integers (whole numbers), floats, characters, texts and many more.

Related course: Complete Python Programming Course & Exercises

Example

Empty list

Lets create an empty list. To define an empty list you should use brackets.
Brackets is what tells Python that the object is a list.

```
list = []
```

Lists can hold both numbers and text. Regardless of contents, they are accessed in the same fashion.

To access a list add the id between the brackets, such as list[0], list[1] and so on.

Define list

An empty list was defined above. Lists can contain all kinds of data.
You can create numeric lists like this:

```
ratings = [ 3,4,6,3,4,6,5 ]
```

Lists can contain strings or characters:

```
ratings = [ 'A','A','B','A','C','A' ]
```

To output simple print them

```
print(ratings)
```

You can interact item by item using a for loop.

Access list items

You can access a list item by using brackets and its index. Python starts counting at zero, that means the first element is zero.

List operations

Lists can be changed with several methods. What are these methods?

To add items to a list, you can use the **append()** method. Call the method on the list, the parameter contains the item to add. Calling append(3) would add 3 to the list. To remove an item from the end of the list, you can use the **pop()** method.

Lists can be accessed like traditional arrays, use the block quotes and index to get an item.

Related course: Complete Python Programming Course & Exercises

Example

Lists can be modified using their methods.
In the example below we create a list and use methods to change the list contents.

Append and pop

Type the program shown below and run it:

```
x = [3,4,5]
x.append(6)
print(x)
x.append(7)
print(x)
x.pop()
print(x)
```

Access items

To access items, simply use the block quotes:

```
x = [3,4,5]

print(x[0])
print(x[1])

print(x[-1])
```

If you are a beginner, then I highly recommend this book.

Exercise

Try the exercises below

1. Given the list y = [6,4,2] add the items 12, 8 and 4.
2. Change the 2nd item of the list to 3.

After completing these continue with the next exercise.

Class(es) and Objects in Python

Python class is concept of *"object oriented programming"*. Python is an object oriented programming language (oop). OOP is a way to build software.

With OOP you can make your program much more organized, scalable, reusable and extensible. The OOP concept can be a bit weird. It can be challenging to grasp, but it's a very powerful concept.

Related course: Complete Python Programming Course & Exercises

Example

Objects

In Python, you can define objects. An object is a collection of methods and variables. Objects live somewhere in the computers memory. They can be manipulated at runtime.

Lets create a theoritcal example, we create an object dog. Creating an object is just one line of code:

```
obj1 = dog()
```

Each object can have variables. The values of those variables are unique to the object. We set object variables (name,age)

```
obj1.name = "Woof"
obj1.age = 5
```

If methods exist for an object, they can be called. The objects unique variables can be used in those methods.
The methods can be used multiple times:

```
obj1.bark()
obj1.bark()
```

In your program you can have multiple objects. Those objects can be of the same type or a different type.

```
obj1 = dog()
obj2 = dog()
obj3 = dog()
obj4 = bird()
```

So how does Python know the type of an object? How does it know which methods and variables exist for a type? They are defined in a class.

What is a constructor in Python?

The constructor is a method that is called when an object is created. This method is defined in the class and can be used to initialize basic variables.

If you create four objects, the class constructor is called four times. Every class has a constructor, but its not required to explicitly define it.

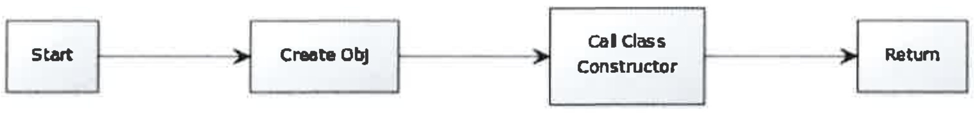
Related course: Complete Python Programming Course & Exercises

Example

Constructor

Each time an object is created a method is called. That methods is named the **constructor**.

The constructor is created with the function **init**. As parameter we write the self keyword, which refers to itself (the object). The process visually is:



Inside the constructor we initialize two variables: legs and arms. Sometimes variables are named properties in the context of object oriented programming. We create one object (bob) and just by creating it, its variables are initialized.

```
class Human:
   def __init__(self):
       self.legs = 2
       self.arms = 2

bob = Human()
print(bob.legs)
```

The newly created object now has the variables set, without you having to define them manually. You could create tens or hundreds of objects without having to set the values each time.

python __init__

The function **init**(self) builds your object. Its not just variables you can set here, you can call class methods too. Everything you need to initialize the object(s).

Lets say you have a class Plane, which upon creation should start flying. There are many steps involved in taking off: accelerating, changing flaps, closing the wheels and so on.

The default actions can be defined in methods. These methods can be called in the constructor.

```
class Plane:
```

Getter and Setter in Python

A class can have one more variables (sometimes called properties). When you create objects each of those objects have unique values for those variables.

Class variables need not be set directly: they can be set using class methods. This is the object orientated way and helps you avoid mistakes.

Related course: Complete Python Programming Course & Exercises

Example

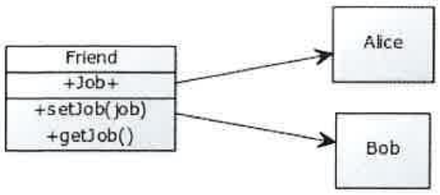
We create a class with a properties. From that class we create several objects.

```
class Friend:
    def __init__(self):
        self.job = "None"

Alice = Friend()
Bob = Friend()
```

These objects do not have the property (job) set. To set it, we could set it directly but that's a bad practice. Instead we create two methods: getJob() and setJob().

```
class Friend:
    def __init__(self):
        self.job = "None"

    def getJob(self):
        return self.job

    def setJob(self, job):
        self.job = job

Alice = Friend()
Bob = Friend()

Alice.setJob("Carpenter")
Bob.setJob("Builder")

print(Bob.job)
print(Alice.job)
```

Two objects are created, both of them have unique values for the property job:



Python Modules and Packages - An Introduction

Modules can have one or more functions. They help you to organize your code. Instead of one long Python file, you can have several files (modules).

A module is a Python file that has functions or classes. A Python program can use one or more modules.

Related course: Complete Python Programming Course & Exercises

Example

What is a module?

There are many modules (sometimes called libraries) available for Python. By using these modules you can code much faster.

Think of them like building blocks, they contain large sets of functions (sometimes classes) that provide you with additional functionality.

Import modules

You can load a module with the **import** keyword.

In the example below we load the *os module*. This is short for operating system, so you can do system tasks.

```
import os
os.system("dir")
```

Using that module we call one of its functions named system (runs a command).

In this case it will simply list the files in the directory (dir command).

There are many many modules available for Python.

Get specific functions from a module

To import a specific function in a module, you can use the line:

```
from module import function
```

There's a module named *time* which has all kind of functionality for time: get the date, hour, minute, second and so on. That's quite a lot of functionality.

Lets say you want the program to wait 2 seconds. If you want, you can import a specific function instead of the whole module.

Inheritance in Python (With Examples)

Inheritance: A class can get the properties and variables of another class. This class is called the super class or parent class.

Inheritances saves you from repeating yourself *(in coding: dont repeat yourself)*, you can define methods once and use them in one or more subclasses.

Related course: Complete Python Programming Course & Exercises

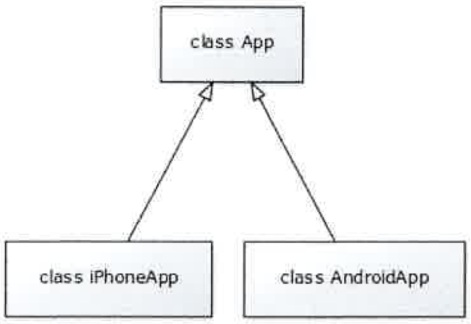
Example

Introduction

You need at least two classes for inheritance to work. Like real life, one will inherit from the other.
The class that inherits from the super class, will get everything. What's everything?

In the case of object oriented programming that means it will get the methods and variables from the super class.

Multiple classes can inherit from the same super class. In such case all of sub classes will get all of the properties and methods of the super class.



How it works

Define two classes, one super class (App) and one sub class (Android). The sub class (Android) inherits from the class App.

First we define the super class. The super class is written just like a normal class, there's nothing special about it except that others will inherit from it. You can give it methods and variables if you want.

```
class App:
    def start(self):
        print('starting')
```

We defined methods and variables in the super class (App), once inherited we can use them in the sub class. Let's create a class (Android) that inherits from the super class.

In the super class we create the method start(). This is just for demonstration purpose, the method will be usable when creating an object with the class Android.

How does Python know a class wants to inherit? The braces after it with the class name.

```
class Android(App):
```

First the normal class name is defined, the super class is defined after that.

Code Example

The example below is a demonstration of inheritance in Python. Python supports multiple inheritance, but in this example we inherit from only one super class.

Complete example below:

```
#!/usr/bin/python

class App:
    def start(self):
       print('starting')

class Android(App):
    def getVersion(self):
       print('Android version')

app = Android()
app.start()
app.getVersion()
```

If you are a beginner, then I highly recommend this book.

Exercises

Try the exercises below:

1. Create a new class that inherits from the class App
2. Try to create a class that inherits from two super classes (multiple inheritance)

Download examples

Back **Next**

Python Modules and Packages - An Introduction Python's Static Methods Demystified

Motoren

JakobGreten edited this page on 6 Oct 2020 · 1 revision

Um einen Motor zu benutzen muss man zuerst ein Motor-Objekt erstellen und dabei den Port angeben. Für den Port C sieht das dann zum Beispiel so aus: `leftMotor = Motor(Port.C)` . Anschließend kann man mit der `run_target` -Methode den Motor um einen bestimmten Winkel drehen oder mit der `run` -Methode den Motor ohne Begrenzung laufen lassen. Um den Motor wieder anzuhalten, sollte man die `stop` -Methode benutzen.

```
#Motor auf Port C
leftMotor = Motor(Port.C)

# Mit einer Zielgeschwindigkeit von 300 Grad pro Sekunde den Motor um 90 Grad zum
Ursprung drehen
leftMotor.run_target(300,90)

#Warte 2 Sekunden
wait(2000)

# Mit einer Zielgeschwindigkeit von 200 Grad pro Sekunde den Motor starten
leftMotor.run(200)

#Warte 2 Sekunden waehrend der Motor laeuft
wait(5000)

#Stoppe den Motor nach 5 Sekunden
leftMotor.stop()
```

Clone this wiki locally

`https://github.com/ai-`

Ultraschallsensor

JakobGreten edited this page on 6 Oct 2020 · 2 revisions

Der Ultraschallsensor misst Entfernungen zwischen 3 und 250cm. Der Messwert wird in Millimeter übergeben.

```
#Ultraschallsensor auf Port 1
ultrasonicSensor =  UltrasonicSensor(Port.S1)

#Gemessene Distanz lesen (in Millimeter)
d=ultrasonicSensor.distance()
```

Display

JakobGreten edited this page on 6 Oct 2020 · 1 revision

Genauso wie die Konsolenausgabe kann das Display genutzt werden um Zwischenausgaben zu machen. Dafür gibt es die Funktion `screen.print()` . Bilder können über die `Image` -Klasse geladen werden und über `screen.load_image` gezeichnet werden. Das Laden der Bilder von der SD-Karte kann etwas dauern, also sollte es möglichst nicht in einer Schleife ausgeführt werden.

```
#Zeige den Text "Hallo" auf dem Display
ev3.screen.print("Hallo")

#Lade eines der von Lego bereitgestellten Bilder.
image1 = Image(ImageFile.WINKING)

# Alternativ kann auch ein png geladen werden
image2 = Image("DemoPic.png")

# Zwischen Laden des Bildes und dem Zeichnen
# sollte am Besten etwas Zeit liegen(wenige Milisekunden reichen)
wait(3000)

# Zeichne das erste Bild
ev3.screen.load_image(image1)

wait(2000)

#Lösche alles was vorher auf dem Display angezeigt wurde
ev3.screen.clear()

# Zeichne das zweite Bild
ev3.screen.load_image(image2)

# Stelle das Statuslicht auf Rot
ev3.light.on(Color.RED)
```

Clone this wiki locally

`https://github.com/ai-`

Buttons

JakobGreten edited this page on 6 Oct 2020 · 1 revision

Die Tasten des EV3 sind sehr nützlich um beispielsweise das Programm zu beenden oder Parameter des Programms während der Laufzeit zu verändern. Die aktivierten Tasten kann man durch `buttons.pressed()` bekommen. Mögliche Buttons sind: `DOWN`, `LEFT`, `CENTER`, `RIGHT`, `UP`.

```
#!/usr/bin/env pybricks-micropython
from pybricks.hubs import EV3Brick
from pybricks.parameters import Button, Color
from pybricks.tools import (wait, print)
from pybricks.media.ev3dev import Image, ImageFile


#Erstelle ein EV3-Objekt
ev3 =EV3Brick()

#Püfe ob der untere Button gedrückt wurde
down_pressed = Button.DOWN in ev3.buttons.pressed()

#Führe eine Schleife so lange aus, bis der untere Button gedrückt wurde
while not Button.DOWN in ev3.buttons.pressed():
    print('Warte auf Button.DOWN')
```