: Python documentation:

Python is a popular programming language.

Python can be used on a server to create web applications.

It is used for:

* web development (server-side),
* software development,
* mathematics,
* system scripting.

What can Python do?

* Python can be used on a server to create web applications.
* Python can be used alongside software to create workflows.
* Python can connect to database systems. It can also read and modify files.
* Python can be used to handle big data and perform complex mathematics.
* Python can be used for rapid prototyping, or for production-ready software development.

Why Python?

* Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
* Python has a simple syntax similar to the English language.
* Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
* Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
* Python can be treated in a procedural way, an object-oriented way or a functional way.

Good to know

Python will be written in a text editor. It is possible to write Python in an Integrated Development Environment, such as Thonny, PyCharm, NetBeans or Eclipse which are particularly useful when managing larger collections of Python files.

Python Syntax compared to other programming languages

* Python was designed for readability and has some similarities to the English language with influence from mathematics.
* Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
* Python relies on indentation, using whitespace, to define scope, such as the scope of loops, functions and classes. Other programming languages often use curly brackets for this purpose.

## Python Install

* To check if you have python installed on a Windows PC, search in the start bar for Python or run the following on the Command Line (cmd.exe):
* C:\Users\*Your Name*>python --version
* To check if you have python installed on a Linux or Mac, then on Linux open the command line or on Mac open the Terminal and type:
* python --version
* If you find that you do not have Python installed on your computer, then you can download it for free from the following website: <https://www.python.org/>
* Python is an interpreted programming language; this means that as a developer you write Python (.py) files in a text editor and then put those files into the python interpreter to be executed.
* The way to run a python file is like this on the command line:
* C:\Users\*Your Name*>python helloworld.py
* Where "helloworld.py" is the name of your python file.
* Let's write our first Python file, called helloworld.py, which can be done in any text editor.
* print("Hello, World!")
* Save your file. Open your command line, navigate to the directory where you saved your file, and run:
* C:\Users\*Your Name*>python helloworld.py
* The output should read:
* Hello, World

## The Python Command Line:

To test a short amount of code in python sometimes it is quickest and easiest not to write the code in a file. This is made possible because Python can be run as a command line itself.

Type the following on the Windows, Mac or Linux command line:

C:\Users\Your Name>python

Or, if the "python" command did not work, you can try "py":

C:\Users\Your Name>py

From there you can write any python, including our hello world example from earlier in the tutorial:

C:\Users\Your Name>python

## >>> print("Hello, World!")

Output: Hello, World!

Or by creating a python file on the server, using the .py file extension, and running it in the Command Line:

C:\Users\*Your Name*>python myfile.py

## **Python Indentation**

Indentation refers to the spaces at the beginning of a code line.

Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.

Python uses indentation to indicate a block of code.

if 5 > 2:  
  print("Five is greater than two!")

Python will give you an error if you skip the indentation.

if 5 > 2:  
print("Five is greater than two!")

The number of spaces is up to you as a programmer, the most common use is four, but it has to be at least one.

if 5 > 2:  
 print("Five is greater than two!")   
if 5 > 2:  
        print("Five is greater than two!")

You have to use the same number of spaces in the same block of code, otherwise Python will give you an error:

if 5 > 2:  
 print("Five is greater than two!")  
        print("Five is greater than two!")

## Comments

Python has commenting capability for the purpose of in-code documentation.

Comments start with a #, and Python will render the rest of the line as a comment:

#This is a comment.  
print("Hello, World!")

Comments can be placed at the end of a line, and Python will ignore the rest of the line:

print("Hello, World!") #This is a comment

A comment does not have to be text that explains the code, it can also be used to prevent Python from executing code:

#print("Hello, World!")  
print("Cheers, Mate!")

## **Multi Line Comments**

Python does not really have a syntax for multi-line comments.

To add a multiline comment, you could insert a # for each line:

#This is a comment  
#written in  
#more than just one line  
print("Hello, World!")

Since Python will ignore string literals that are not assigned to a variable, you can add a multiline string (triple quotes) in your code and place your comment inside it.

"""  
This is a comment  
written in  
more than just one line  
"""  
print("Hello, World!")

As long as the string is not assigned to a variable, Python will read the code, but then ignore it, and you have made a multiline comment. Python has no command for declaring a variable.

Variables:

Variables are the containers for storing data.

A variable is created the moment you first assign a value to it.

x = 5  
y = "John"  
print(x)  
print(y)

Variables do not need to be declared with any particular *type*, and can even change type after they have been set.

x = 4       # x is of type int  
x = "Sally" # x is now of type str  
print(x)

## Casting

## If you want to specify the data type of a variable, this can be done with casting.

x = str(3)    # x will be '3'  
y = int(3)    # y will be 3  
z = float(3)  # z will be 3.0

## Get the Type:

You can get the data type of a variable with the type() function.

x = 5  
y = "John"  
print(type(x))  
print(type(y))

String variables can be declared either by using single or double quotes:

x = "John"  
# is the same as  
x = 'John'

## Case-Sensitive:

Variable names are case-sensitive.

This will create 2 variables.

a = 4  
A = "Sally"  
#A will not overwrite a.

## Variable Names:

A variable can have a short name (like x and y) or a more descriptive name (age, car name, total volume).

Rules for Python variables:

. A variable name must start with a letter or the underscore character

. A variable name cannot start with a number

* A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and \_)
* Variable names are case-sensitive (age, Age and AGE are three different variables)
* Legal variable names:

myvar = "John"  
 my\_var = "John"  
 \_my\_var = "John"  
 myVar = "John"  
 MYVAR = "John"  
 myvar2 = "John"

Example

Illegal variable names:

2myvar = "John"  
my-var = "John"  
my var = "John"

## Multi Words Variable Names :

Variable names with more than one word can be difficult to read.

There are several techniques you can use to make them more readable:

## Camel Case

Each word, except the first, starts with a capital letter:

myVariableName ="John"

## Pascal Case

Each word starts with a capital letter:

MyVariableName = "John"

## Snake Case

Each word is separated by an underscore character:

my\_variable\_name ="John"

# Python Variables - Assign Multiple Values:

## Many Values to Multiple Variables

Python allows you to assign values to multiple variables in one line:

x, y, z = "Orange", "Banana", "Cherry"  
print(x)  
print(y)  
print(z)

## One Value to Multiple Variables

And you can assign the same value to multiple variables in one line:

x = y = z = "Orange"  
print(x)  
print(y)  
print(z)

## Unpack a Collection:

If you have a collection of values in a list, tuple etc. Python allows you to extract the values into variables. This is called unpacking.

Unpack a list:

fruits = ["apple", "banana", "cherry"]  
x, y, z = fruits  
print(x)  
print(y)  
print(z)

## Output Variables:

The Python print() function is often used to output variables.

x = "Python is awesome"  
print(x)

In the print() function to output multiple variables, separated by a comma:

x = "Python"  
y = "is"  
z = "awesome"  
print(x, y, z)

You can also use the + operator to output multiple variables:

x = "Python "  
y = "is "  
z = "awesome"  
print(x + y + z)

For numbers, the + character works as a mathematical operator:

x = 5

y = 10  
print(x + y)

In the print() function, when you try to combine a string and a number with the + operator, Python will give you an error:

x = 5  
y = "John"  
print(x + y)

The best way to output multiple variables in the print() function is to separate them with commas, which even support different data types:

x = 5  
y = "John"  
print(x, y)

## Global Variables

Variables that are created outside of a function (as in all of the examples above) are known as global variables.

Global variables can be used by everyone, both inside of functions and outside.

Create a variable outside of a function and use it inside the function.

x = "awesome"  
  
def myfunc():  
  print("Python is " + x)  
  
myfunc()

If you create a variable with the same name inside a function, this variable will be local, and can only be used inside the function. The global variable with the same name will remain as it was, global and with the original value.

x = "awesome"  
  
def myfunc():  
  x = "fantastic"  
  print("Python is " + x)  
  
myfunc()  
  
print("Python is " + x)

## The global Keyword

Normally, when you create a variable inside a function, that variable is local, and can only be used inside that function.

To create a global variable inside a function, you can use the global keyword.

def myfunc():  
  global x  
  x = "fantastic"  
  
myfunc()  
  
print("Python is " + x)

Also, use the global keyword if you want to change a global variable inside a function.

x = "awesome"  
  
def myfunc():  
  global x  
  x = "fantastic"  
  
myfunc()  
  
print("Python is " + x)

# **Python Strings:**

## Strings:

Strings in python are surrounded by either single quotation marks, or double quotation marks.

'hello' is the same as "hello".

You can display a string literal with the print() function:

print("Hello")  
print('Hello')

## Assign String to a Variable:

Assigning a string to a variable is done with the variable name followed by an equal sign and the string:

a = "Hello"  
print(a)

## Multiline Strings:

You can assign a multiline string to a variable by using three quotes.

a = """Lorem ipsum dolor sit amet,  
consectetur adipiscing elit,  
sed do eiusmod tempor incididunt  
ut labore et dolore magna aliqua."""  
print(a)

Or three single quotes:

a = '''Lorem ipsum dolor sit amet,  
consectetur adipiscing elit,  
sed do eiusmod tempor incididunt  
ut labore et dolore magna aliqua.'''  
print(a)

## Strings are Arrays:

Like many other popular programming languages, strings in Python are arrays of bytes representing unicode characters.

However, Python does not have a character data type, a single character is simply a string with a length of 1.

Square brackets can be used to access elements of the string.

a = "Hello, World!"  
print(a[1])

## Looping Through a String:

Since strings are arrays, we can loop through the characters in a string, with a for loop.

To get the length of a string, use the len() function.

a = "Hello, World!"  
print(len(a))

## Check String

To check if a certain phrase or character is present in a string, we can use the keyword.

txt = "The best things in life are free!"  
print("free" in txt)

output: true

Use it in an if statement:

txt = "The best things in life are free!"  
if "free" in txt:  
  print("Yes, 'free' is present.")

## Check if NOT

To check if a certain phrase or character is NOT present in a string, we can use the keyword not in.

txt = "The best things in life are free!"  
print("expensive" not in txt)

output: true

txt = "The best things in life are free!"

if "expensive" not in txt:  
  print("No, 'expensive' is NOT present.")

Use it in an if statement:

### Example

print only if "expensive" is NOT present:

txt = "The best things in life are free!"

if "expensive" not in txt:  
  print("No, 'expensive' is NOT present.")

output: No, 'expensive' is NOT present

## Negative Indexing:

Use negative indexes to start the slice from the end of the string.

Get the characters:

From: "o" in "World!" (Position -5)

To, but not included: "d" in "World!" (Position -2):

b = "Hello, World!"  
print(b[-5:-2])

output: orl

# Python - Modify Strings:

Python has a set of built-in methods that you can use on strings.

## Upper Case

a = "Hello, World!"  
print(a.upper())

output: HELLO,WORLD!

## Lower Case

a = "Hello, World!"  
print(a.lower())

output: hello,world!

## *Remove Whitespace*

Whitespace is the space before and/or after the actual text, and very often you want to remove this space.

a = " Hello, World! "  
print(a.strip()) # returns "Hello, World!"

output:Hello,World!

## Replace string

a = "Hello, World!"  
print(a.replace("H", "J"))

output: Jello, World!

## Split String

The split() method returns a list where the text between the specified separator becomes the list items.

a = "Hello, World!"  
print(a.split(",")) # returns ['Hello', ' World!']

output: ['Hello', ' World!']

# Python - String Concatenation

## String Concatenation

To concatenate, or combine, two strings you can use the + operator.

a = "Hello"  
b = "World"  
c = a + b  
print(c)

output: Hello World

To add a space between them, add a " ":

a = "Hello"  
b = "World"  
c = a + " " + b  
print(c)

output: Hello World

# Python - Format - Strings

## **String Format**

As we learned in the Python Variables chapter, we cannot combine strings and numbers like this:

age = 36  
txt = "My name is John, I am " + age  
print(txt)

output: Traceback (most recent call last):  
  File "demo\_string\_format\_error.py", line 2, in <module>  
    txt = "My name is John, I am " + age  
TypeError: must be str, not int

But we can combine strings and numbers by using the format() method!

The format() method takes the passed arguments, formats them, and places them in the string where the placeholders {} are:

age = 36  
txt = "My name is John, and I am {}"  
print(txt.format(age))

output: My name is John, and I am 36

The format() method takes unlimited number of arguments, and are placed into the respective placeholders:

quantity = 3  
itemno = 567  
price = 49.95  
myorder = "I want {} pieces of item {} for {} dollars."  
print(myorder.format(quantity, itemno, price))

output: I want 3 pieces of item 567 for 49.95 dollars.

You can use index numbers {0} to be sure the arguments are placed in the correct placeholders:

quantity = 3  
itemno = 567  
price = 49.95  
myorder = "I want to pay {2} dollars for {0} pieces of item {1}."  
print(myorder.format(quantity, itemno, price))

output: I want to pay 49.95 dollars for 3 pieces of item 567

# Python - Escape Characters:

## Escape Character

To insert characters that are illegal in a string, use an escape character.

An escape character is a backslash \ followed by the character you want to insert.

An example of an illegal character is a double quote inside a string that is surrounded by double quotes:

txt = "We are the so-called "Vikings" from the north."

Output: File "demo\_string\_escape\_error.py", line 1  
    txt = "We are the so-called "Vikings" from the north."  
                                       ^  
SyntaxError: invalid syntax

To fix this problem, use the escape character .

txt = "We are the so-called \"Vikings\" from the north."

Output: We are the so-called "Vikings" from the north.

## **Escape Characters**

Other escape characters used in Python:

|  |  |  |
| --- | --- | --- |
| **Code** | **Result** | **Try it** |
| \' | Single Quote | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_string_escape2) |
| \\ | Backslash | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_string_backslash) |
| \n | New Line | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_string_newline) |
| \r | Carriage Return | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_string_r) |
| \t | Tab | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_string_t) |
| \b | Backspace | [Try it »](https://www.w3schools.com/python/showpython.asp?filename=demo_string_b) |
| \f | Form Feed |  |
| \ooo | Octal value | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_string_octal) |
| \xhh | Hex value | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_string_hex) |

# Python Operators

## Python Operators

Operators are used to perform operations on variables and values.

In the example below, we use the + operator to add together two values:

print(10 + 5)

output:15

Python divides the operators in the following groups:

* Arithmetic operators
* Assignment operators
* Comparison operators
* Logical operators
* Identity operators
* Membership operators
* Bitwise operators

## Python Arithmetic Operators

Arithmetic operators are used with numeric values to perform common mathematical operations:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Name** | **Example** | **Try it** |
| + | Addition | x + y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_add) |
| - | Subtraction | x - y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_sub) |
| \* | Multiplication | x \* y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_mult) |
| / | Division | x / y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_div) |
| % | Modulus | x % y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_mod) |
| \*\* | Exponentiation | x \*\* y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_exp) |
| // | Floor division | x // y |  |

Python assignment operators:

Assignment operators are used to assign values to variables:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Example** | **Same As** |  |
| = | x = 5 | x = 5 |  |
| += | x += 3 | x = x + 3 |  |
| -= | x -= 3 | x = x - 3 |  |
| \*= | x \*= 3 | x = x \* 3 |  |
| /= | x /= 3 | x = x / 3 |  |
| %= | x %= 3 | x = x % 3 |  |
| //= | x //= 3 | x = x // 3 |  |
| \*\*= | x \*\*= 3 | x = x \*\* 3 |  |
| &= | x &= 3 | x = x & 3 |  |
| |= | x |= 3 | x = x | 3 |  |
| ^= | x ^= 3 | x = x ^ 3 |  |
| >>= | x >>= 3 | x = x >> 3 |  |
| <<= | x <<= 3 | x = x << 3 |  |

## Python Comparison Operators

Comparison operators are used to compare two values:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Name** | **Example** | **Try it** |
| == | Equal | x == y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_compare1) |
| != | Not equal | x != y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_compare2) |
| > | Greater than | x > y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_compare4) |
| < | Less than | x < y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_compare5) |
| >= | Greater than or equal to | x >= y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_compare6) |
| <= | Less than or equal to | x <= y |  |

## Python Logical Operators

Logical operators are used to combine conditional statements:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Description** | **Example** | **Try it** |
| and | Returns True if both statements are true | x < 5 and  x < 10 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_logical1) |
| or | Returns True if one of the statements is true | x < 5 or x < 4 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_logical2) |
| not | Reverse the result, returns False if the result is true | not(x < 5 and x < 10  ) |  |

## Python Identity Operators

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Description** | **Example** | **Try it** |
| is | Returns True if both variables are the same object | x is y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_identity1) |
| is not | Returns True if both variables are not the same object | x is not y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_identity2) |

## Python Membership Operators

Membership operators are used to test if a sequence is presented in an object:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Description** | **Example** | **Try it** |
| in | Returns True if a sequence with the specified value is present in the object | x in y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_membership1) |
| not in | Returns True if a sequence with the specified value is not present in the object | x not in y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_membership2) |

## Python Bitwise Operators

Bitwise operators are used to compare (binary) numbers:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Name** | **Description** |
| **&** | **AND** | **Sets each bit to 1 if both bits are 1** |
| **|** | **OR** | **Sets each bit to 1 if one of two bits is 1** |
| **^** | **XOR** | **Sets each bit to 1 if only one of two bits is 1** |
| **~** | **NOT** | **Inverts all the bits** |
| **<<** | **Zero fill left shift** | **Shift left by pushing zeros in from the right and let the**  **leftmost bits fall off** |
| **>>** | **Signed right shift** | **Shift right by pushing copies of the leftmost bit in from**  **the left, and let the rightmost bits fall off** |

## Built-in Data Types:

In programming, data type is an important concept.

Variables can store data of different types, and different types can do different things.

Python has the following data types built-in by default, in these categories:

|  |  |
| --- | --- |
| Text Type: | str |
| Numeric Types: | int, float, complex |
| Sequence Types: | list, tuple, range |
| Mapping Type: | dict |
| Set Types: | set, frozenset |
| Boolean Type: | bool |
| Binary Types: | bytes, bytearray, memoryview |
| None Type: | NoneType |

## **Python Conditions and If statements**

Python supports the usual logical conditions from mathematics:

* Equals: a == b
* Not Equals: a != b
* Less than: a < b
* Less than or equal to: a <= b
* Greater than: a > b
* Greater than or equal to: a >= b

These conditions can be used in several ways, most commonly in "if statements" and loops.

An "if statement" is written by using the if keyword.

Ex: a = 33  
b = 200  
if b > a:  
  print("b is greater than a"

Ouput: b is greater than a

## **Indentation:**

Python relies on indentation (whitespace at the beginning of a line) to define scope in the code. Other programming languages often use curly brackets for this purpose.

a = 33  
b = 200  
if b > a:  
print("b is greater than a") # you will get an error

output: File "demo\_if\_error.py", line 4  
     print("b is greater than a")  
         ^

IndentationError: expected an indented block

## Elif:

The elif keyword is pythons’ way of saying "if the previous conditions were not true, then try this condition".

Ex: a = 33  
b = 33  
if b > a:  
  print("b is greater than a")  
elif a == b:  
  print("a and b are equal")

Output: a and b are equal

## Else

The else keyword catches anything which isn't caught by the preceding conditions.

a = 200  
b = 33  
if b > a:  
  print("b is greater than a")  
elif a == b:  
  print("a and b are equal")  
else:  
  print("a is greater than b")

output: a is greater than b.

You can also have an else without the elif:

a = 200  
b = 33  
if b > a:  
  print("b is greater than a")  
else:  
  print("b is not greater than a")

output: b is not greater than a

## Shorthand If:

If you have only one statement to execute, you can put it on the same line as the if statement.

if a > b: print("a is greater than b")

output: a is greater than b.

## Shorthand If ... Else

If you have only one statement to execute, one for if, and one for else, you can put it all on the same line:

a = 2  
b = 330  
print("A") if a > b else print("B")

output: **B**

## And:

The and keyword is a logical operator, and is used to combine conditional statements:

a = 200  
b = 33  
c = 500  
if a > b and c > a:  
  print("Both conditions are True")

output: both conditions are true.

## Or

The or keyword is a logical operator, and is used to combine conditional statements:

a = 200  
b = 33  
c = 500  
if a > b or a > c:  
  print("At least one of the conditions is True")

output: at least one of the cond’s is true

## Nested If

You can have if statements inside if statements, this is called nested if statements.

x = 41  
  
if x > 10:  
  print("Above ten,")  
  if x > 20:  
    print("and also above 20!")  
  else:  
    print("but not above 20.")

## The pass Statement

if statements cannot be empty, but if you for some reason have an if statement with no content, put in the pass statement to avoid getting an error.

a = 33  
b = 200  
  
if b > a:  
  pass

# having an empty if statement like this, would raise an error without the pass statement

## Python Loops

Python has two primitive loop commands:

* while loops
* for loops

## The while Loop

With the while loop we can execute a set of statements as long as a condition is true.

i = 1  
while i < 6:  
  print(i)  
  i += 1

output:1 2 3 4 5

## The break Statement

With the break statement we can stop the loop even if the while condition is true:

i = 1  
while i < 6:  
  print(i)  
  if i == 3:  
    break  
  i += 1

output: 1 2 3

## The continue Statement

With the continue statement we can stop the current iteration, and continue with the next:

i = 0  
while i < 6:  
  i += 1  
  if i == 3:  
    continue  
  print(i)

output:1 2 3 4 5 6

## The else Statement

With the else statement we can run a block of code once when the condition no longer is true:

i = 1  
while i < 6:  
  print(i)  
  i += 1  
else:  
  print("i is no longer less than 6")

output:1 2 3 4 5 I is no longer less than 6

## Python For Loops

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.

With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.

fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
  print(x)

## Looping Through a String

Even strings are iterable objects, they contain a sequence of characters:

for x in "banana":  
  print(x)

output: b a n a n a

## The break Statement

With the break statement we can stop the loop before it has looped through all the items:

fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
  print(x)  
  if x == "banana":  
    break

output: apple

banana

fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
  if x == "banana":  
    break  
  print(x)

output: apple

## The continue Statement

With the continue statement we can stop the current iteration of the loop, and continue with the next:

fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
  if x == "banana":  
    continue  
  print(x)

output: apple

cherry

## The range() Function

To loop through a set of code a specified number of times, we can use the range() function,

The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

for x in range(6):  
  print(x)

output:0 1 2 3 4 5

for x in range(2, 30, 3):  
  print(x)

output: 2 5 8 11 14 17 20 23 26 29

## Else in For Loop

The else keyword in a for loop specifies a block of code to be executed when the loop is finished:

for x in range(6):  
  print(x)  
else:  
  print("Finally finished!")

output: 0 1 2 3 4 5 finally finished

for x in range(6):  
  if x == 3: break  
  print(x)  
else:  
  print("Finally finished!")

output:1 2 3

## Nested Loops

A nested loop is a loop inside a loop.

The "inner loop" will be executed one time for each iteration of the "outer loop":

adj = ["red", "big", "tasty"]  
fruits = ["apple", "banana", "cherry"]  
  
for x in adj:  
  for y in fruits:  
    print(x, y)

output: red apple, red banana, red cherry

big apple, banana, big cherry

tasty apple, tasty banana, tasty cherry

## The pass Statement

for loops cannot be empty, but if you for some reason have a for loop with no content, put in the pass statement to avoid getting an error.

for x in [0, 1, 2]:  
  pass

# having an empty for loop like this, would raise an error without the pass statement.

Python functions:

A function is a block of code which only runs when it is called.

You can pass data, known as parameters, into a function.

A function can return data as a result.

## Creating a Function

In Python a function is defined using the def keyword:

def my\_function():  
  print("Hello from a function")

## Calling a Function

To call a function, use the function name followed by parenthesis:

def my\_function():  
  print("Hello from a function")  
  
**my\_function()**

**output: Hello from a functions**

## Number of Arguments

By default, a function must be called with the correct number of arguments. Meaning that if your function expects 2 arguments, you have to call the function with 2 arguments, not more, and not less.

def my\_function(fname, lname):  
  print(fname + " " + lname)  
  
my\_function("Emil", "Refsnes")

output: Emil,Refsnes

If you try to call the function with 1 or 3 arguments, you will get an error:

def my\_function(fname, lname):  
  print(fname + " " + lname)  
  
my\_function("Emil")

Traceback (most recent call last):

File "demo\_function\_args\_error.py", line 4, in <module>

my\_function("Emil")

TypeError: my\_function() missing 1 required positional argument: 'lname'

## Arbitrary Arguments, \*args

If you do not know how many arguments that will be passed into your function, add a \* before the parameter name in the function definition.

This way the function will receive a tuple of arguments, and can access the items accordingly:

def my\_function(\*kids):  
  print("The youngest child is " + kids[2])  
  
my\_function("Emil", "Tobias", "Linus")

output:The youngest child is linus.

## Keyword Arguments

You can also send arguments with the key = value syntax.

This way the order of the arguments does not matter.

## Default Parameter Value

The following example shows how to use a default parameter value.

If we call the function without argument, it uses the default value:

def my\_function(**country = "Norway"**):  
  print("I am from " + country)  
  
my\_function("Sweden")  
my\_function("India")  
my\_function()  
my\_function("Brazil")

I am from Sweden

I am from India

I am from Norway

I am from Brazil

## Passing a List as an Argument

You can send any data types of argument to a function (string, number, list, dictionary etc.), and it will be treated as the same data type inside the function.

E.g. if you send a List as an argument, it will still be a List when it reaches the function:

def my\_function(food):  
  for x in food:  
    print(x)  
  
fruits = ["apple", "banana", "cherry"]  
  
my\_function(fruits)

output:apple

banana

cherry

## Return Values

To let a function, return a value, use the return statement:

def my\_function(x):  
  **return 5 \* x**  
print(my\_function(3))  
print(my\_function(5))  
print(my\_function(9))

output:15

25

45

## The pass Statement

function definitions cannot be empty, but if you for some reason have a function definition with no content, put in the pass statement to avoid getting an error.

def myfunction():  
  pass

def myfunction():  
  pass

# having an empty function definition like this, would raise an error without the pass statement

## Recursion

Python also accepts function recursion, which means a defined function can call itself.

Recursion is a common mathematical and programming concept. It means that a function calls itself. This has the benefit of meaning that you can loop through data to reach a result.

The developer should be very careful with recursion as it can be quite easy to slip into writing a function which never terminates, or one that uses excess amounts of memory or processor power. However, when written correctly recursion can be a very efficient and mathematically elegant approach to programming.

In this example, tri\_recursion() is a function that we have defined to call itself ("recurse"). We use the k variable as the data, which decrements (-1) every time we recurse. The recursion ends when the condition is not greater than 0 (i.e. when it is 0).

To a new developer it can take some time to work out how exactly this works, best way to find out is by testing and modifying it

def tri\_recursion(k):  
  if(k > 0):  
    result = k + tri\_recursion(k - 1)  
    print(result)  
  else:  
    result = 0  
  return result  
  
print("\n\nRecursion Example Results")  
tri\_recursion(6)

1 3 6 10 15 21

## Python Classes/Objects

Python is an object-oriented programming language.

Almost everything in Python is an object, with its properties and methods.

A Class is like an object constructor, or a "blueprint" for creating objects.

## Create a Class

To create a class, use the keyword class:

class MyClass:  
  x = 5

<class '\_\_main\_\_.MyClass'>

## Create Object

Now we can use the class named MyClass to create objects:

p1 = MyClass()  
print(p1.x)

output: 5

## The init() Function

The examples above are classes and objects in their simplest form, and are not really useful in real life applications.

To understand the meaning of classes we have to understand the built-in \_init() function.

All classes have a function called init(), which is always executed when the class is being initiated.

Use the init() function to assign values to object properties, or other operations that are necessary to do when the object is being created:

class Person:  
  def \_\_init\_\_(self, name, age):  
    self.name = name  
    self.age = age  
  
p1 = Person("John", 36)  
  
print(p1.name)  
print(p1.age)

output: john

36

## The str Function:

The str() function controls what should be returned when the class object is represented as a string.

If the str() function is not set, the string representation of the object is returned:

class Person:  
  def \_\_init\_\_(self, name, age):  
    self.name = name  
    self.age = age  
  
p1 = Person("John", 36)  
  
print(p1)

output: John(36)

The string representation of an obj with the str function.

class Person:  
  def \_\_init\_\_(self, name, age):  
    self.name = name  
    self.age = age  
  
  def \_\_str\_\_(self):  
    return f"{self.name}({self.age})"  
  
p1 = Person("John", 36)  
  
print(p1)

output: John(36)

## Object Methods

Objects can also contain methods. Methods in objects are functions that belong to the object.

class Person:  
  def \_\_init\_\_(self, name, age):  
    self.name = name  
    self.age = age  
  
  def myfunc(self):  
    print("Hello my name is " + self.name)  
  
p1 = Person("John", 36)  
p1.myfunc()

output: Hello my name is Jonh

## **The self-Parameter**

The self-parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.

It does not have to be named self , you can call it whatever you like, but it has to be the first parameter of any function in the class:

class Person:  
  def \_\_init\_\_(mysillyobject, name, age):  
    mysillyobject.name = name  
    mysillyobject.age = age  
  
  def myfunc(abc):  
    print("Hello my name is " + abc.name)  
  
p1 = Person("John", 36)  
p1.myfunc()

output: Hello my name is John

## Modify Object Properties

You can modify properties on objects like this:

p1.age = 40

output: 40

## Delete Object Properties

You can delete properties on objects by using the del keyword:

del p1.age

output: Traceback (most recent call last):  
  File "demo\_class7.py", line 13, in <module>  
    print(p1.age)  
AttributeError: 'Person' object has no attribute 'age'

## **Delete Objects**

You can delete objects by using the del keyword:

del p1

output: Traceback (most recent call last):  
  File "demo\_class8.py", line 13, in <module>  
    print(p1)  
NameError: 'p1' is not defined

## **The pass Statement**

class definitions cannot be empty, but if you for some reason have a class definition with no content, put in the pass statement to avoid getting an error.

class Person:  
  pass

output: # having an empty class definition like this, would raise an error without the pass statement

## Python Inheritance

Inheritance allows us to define a class that inherits all the methods and properties from another class.

**Parent class** is the class being inherited from, also called base class.

**Child class** is the class that inherits from another class, also called derived class.

## Create a Parent Class

Any class can be a parent class, so the syntax is the same as creating any other class:

Create a class named person with first name, last name properties and aprint method.

class Person:  
  def \_\_init\_\_(self, fname, lname):  
    self.firstname = fname  
    self.lastname = lname  
  
  def printname(self):  
    print(self.firstname, self.lastname)  
  
#Use the Person class to create an object, and then execute the printname method:  
  
x = Person("John", "Doe")  
x.printname()

output: John Doe

## Create a Child Class

To create a class that inherits the functionality from another class, send the parent class as a parameter when creating the child class:

Create a class named student, which will inherit the properties and methods from the person class.

class Student(Person):  
  pass

use the pass keyword when you do not want to add any other properties and methods to the class.

Now the Student class has the same properties and methods as the Person class.

x = Student("Mike", "Olsen")  
x.printname()

output: Mike Oslen

## Add the \_\_init\_\_() Function

So far we have created a child class that inherits the properties and methods from its parent.

We want to add the \_\_init\_\_() function to the child class (instead of the pass keyword).

Add the init function to the student class.

class Student(Person):  
  def \_\_init\_\_(self, fname, lname):  
    #add properties etc.

When you add the \_\_init\_\_() function, the child class will no longer inherit the parent's \_\_init\_\_() function.

class Student(Person):  
  def \_\_init\_\_(self, fname, lname):  
    Person.\_\_init\_\_(self, fname, lname)

output: Mike Oslen

Now we have successfully added the \_\_init\_\_() function, and kept the inheritance of the parent class, and we are ready to add functionality in the \_\_init\_\_() function.

## Use the super() Function

Python also has a super() function that will make the child class inherit all the methods and properties from its parent:

class Student(Person):  
  def \_\_init\_\_(self, fname, lname):  
    super().\_\_init\_\_(fname, lname)

output: Mike oslen

By using the super() function, you do not have to use the name of the parent element, it will automatically inherit the methods and properties from its parent.

## **Add Properties**

Add a property called graduationyear to the student class.

class Student(Person):  
  def \_\_init\_\_(self, fname, lname):  
    super().\_\_init\_\_(fname, lname)  
    self.graduationyear = 2019

output: 2019

In the example below, the year 2019 should be a variable, and passed into the Student class when creating student objects. To do so, add another parameter in the \_\_init\_\_() function:

class Student(Person):  
  def \_\_init\_\_(self, fname, lname, year):  
    super().\_\_init\_\_(fname, lname)  
    self.graduationyear = year  
  
x = Student("Mike", "Olsen", 2019)

output: 2019

## **Add Methods**

Add a method welcome method to the student class.

class Student(Person):  
  def \_\_init\_\_(self, fname, lname, year):  
    super().\_\_init\_\_(fname, lname)  
    self.graduationyear = year  
  
  def welcome(self):  
    print("Welcome", self.firstname, self.lastname, "to the class of", self.graduationyear)

output: welcome mike Olsen to the class of 2019.

If you add a method in the child class with the same name as a function in the parent class, the inheritance of the parent method will be overridden.