What is Python?

Python is a popular programming language. It was created by Guido van Rossum and released in 1991.

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.

Creating Variables

Python has no command for declaring a variable.

A variable can have a short name (like x and y) or a more descriptive name (age, carname, 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)

Variable names are case-sensitive.

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

### Example

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.

### Example

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.

### Example

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.

Example

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

## Many Values to Multiple Variables

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

### Example

x, y, z = "Orange", 5, 2.3

print(x)  
print(y)  
print(z)

## One Value to Multiple Variables

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

### Example

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 extract the values into variables. This is called unpacking.

### Example

Unpack a list:

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

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.

### Example

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

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

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

### Example

To change the value of a global variable inside a function, refer to the variable by using the global keyword:

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

Python Data Types

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 |

|  |  |  |
| --- | --- | --- |
| **Example** | **Data Type** |  |
| x = "Hello World" | str |  |
| x = 20 | int |  |
| x = 20.5 | float |  |
| x = 1j | complex |  |
| x = ["apple", "banana", "cherry"] | list |  |
| x = ("apple", "banana", "cherry") | tuple |  |
| x = range(6) | range |  |
| x = {"name" : "John", "age" : 36} | dict |  |
| x = {"apple", "banana", "cherry"} | set |  |
| x = frozenset({"apple", "banana", "cherry"}) | frozenset |  |
| x = True | bool |  |
| x = b"Hello" | bytes |  |
| x = bytearray(5) | bytearray |  |
| x = memoryview(bytes(5)) | memoryview |  |

## Setting the Specific Data Type

If you want to specify the data type, you can use the following constructor functions:

|  |  |  |
| --- | --- | --- |
| **Example** | **Data Type** |  |
| x = str("Hello World") | str |  |
| x = int(20) | int |  |
| x = float(20.5) | float |  |
| x = complex(1j) | complex |  |
| x = list(("apple", "banana", "cherry")) | list |  |
| x = tuple(("apple", "banana", "cherry")) | tuple |  |
| x = range(6) | range |  |
| x = dict(name="John", age=36) | dict |  |
| x = set(("apple", "banana", "cherry")) | set |  |
| x = frozenset(("apple", "banana", "cherry")) | frozenset |  |
| x = bool(5) | bool |  |
| x = bytes(5) | bytes |  |
| x = bytearray(5) | bytearray |  |
| x = memoryview(bytes(5)) | memoryview |  |

# Python Lists

Lists are used to store multiple items in a single variable.

Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are [Tuple](https://www.w3schools.com/python/python_tuples.asp), [Set](https://www.w3schools.com/python/python_sets.asp), and [Dictionary](https://www.w3schools.com/python/python_dictionaries.asp), all with different qualities and usage.

Lists are created using square brackets:

### Example

Create a List:

thislist = ["apple", "banana", "cherry"]  
print(thislist)

## List Items

List items are ordered, changeable, and allow duplicate values.

List items are indexed, the first item has index [0], the second item has index [1] etc.

## Ordered

When we say that lists are ordered, it means that the items have a defined order, and that order will not change.

If you add new items to a list, the new items will be placed at the end of the list.

## Changeable

The list is changeable, meaning that we can change, add, and remove items in a list after it has been created.

## Allow Duplicates

Since lists are indexed, lists can have items with the same value:

### Example

Lists allow duplicate values:

thislist = ["apple", "banana", "cherry", "apple", "cherry"]  
print(thislist)

A list can contain different data types:

### Example

A list with strings, integers and boolean values:

list1 = ["abc", 34, True, 40, "male"]

## The list() Constructor

It is also possible to use the list() constructor when creating a new list.

### Example

Using the list() constructor to make a List:

thislist = list(("apple", "banana", "cherry")) # note the double round-brackets  
print(thislist)

## Extend List

To append elements from another list to the current list, use the extend() method.

### Example

Add the elements of tropical to thislist:

thislist = ["apple", "banana", "cherry"]  
tropical = ["mango", "pineapple", "papaya"]  
thislist.extend(tropical)  
print(thislist)

## Add Any Iterable

The extend() method does not have to append lists, you can add any iterable object (tuples, sets, dictionaries etc.).

### Example

Add elements of a tuple to a list:

thislist = ["apple", "banana", "cherry"]  
thistuple = ("kiwi", "orange")  
thislist.extend(thistuple)  
print(thislist)

## Clear the List

The clear() method empties the list.

The list still remains, but it has no content.

### Example

Clear the list content:

thislist = ["apple", "banana", "cherry"]  
thislist.clear()  
print(thislist)

The del keyword can also delete the list completely.

### Example

Delete the entire list:

thislist = ["apple", "banana", "cherry"]  
del thislist

## Loop Through a List

You can loop through the list items by using a for loop:

### Example

Print all items in the list, one by one:

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

## Loop Through the Index Numbers

You can also loop through the list items by referring to their index number.

Use the range() and len() functions to create a suitable iterable.

### Example

Print all items by referring to their index number:

thislist = ["apple", "banana", "cherry"]  
for i in range(len(thislist)):  
  print(thislist[i])

The iterable created in the example above is [0, 1, 2].

## List Comprehension

List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.

Example:

Based on a list of fruits, you want a new list, containing only the fruits with the letter "a" in the name.

Without list comprehension you will have to write a for statement with a conditional test inside:

### Example

fruits = ["apple", "banana", "cherry", "kiwi", "mango"]  
newlist = []  
  
for x in fruits:  
  if "a" in x:  
    newlist.append(x)  
  
print(newlist)

With list comprehension you can do all that with only one line of code:

### Example

fruits = ["apple", "banana", "cherry", "kiwi", "mango"]  
  
newlist = [x for x in fruits if "a" in x]  
  
print(newlist)

## The Syntax

newlist = [expression for item in iterable if condition == True]

The return value is a new list, leaving the old list unchanged.

### Condition

The condition is like a filter that only accepts the items that valuate to True.

### Example

Only accept items that are not "apple":

newlist = [x for x in fruits if x != "apple"]

The condition if x != "apple"  will return True for all elements other than "apple", making the new list contain all fruits except "apple".

The condition is optional and can be omitted:

### Example

With no if statement:

newlist = [x for x in fruits]

### Iterable

The iterable can be any iterable object, like a list, tuple, set etc.

### Example

You can use the range() function to create an iterable:

newlist = [x for x in range(10)]

Same example, but with a condition:

### Example

Accept only numbers lower than 5:

newlist = [x for x in range(10) if x < 5]

### Expression

The expression is the current item in the iteration, but it is also the outcome, which you can manipulate before it ends up like a list item in the new list:

### Example

Set the values in the new list to upper case:

newlist = [x.upper() for x in fruits]

You can set the outcome to whatever you like:

### Example

Set all values in the new list to 'hello':

newlist = ['hello' for x in fruits]

The expression can also contain conditions, not like a filter, but as a way to manipulate the outcome:

### Example

Return "orange" instead of "banana":

newlist = [x if x != "banana" else "orange" for x in fruits]

The expression in the example above says:

"Return the item if it is not banana, if it is banana return orange".

## Looping Using List Comprehension

List Comprehension offers the shortest syntax for looping through lists:

### Example

A short hand for loop that will print all items in a list:

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

# Python - Sort Lists

List objects have a sort() method that will sort the list alphanumerically, ascending, by default:

thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]  
thislist.sort()  
print(thislist)

By default the sort() method is case sensitive, resulting in all capital letters being sorted before lower case letters:

To sort descending, use the keyword argument reverse = True:

thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]  
thislist.sort(reverse = True)  
print(thislist)

## Customize Sort Function

You can also customize your own function by using the keyword argument key = function.

### Example

Sort the list based on how close the number is to 50:

def myfunc(n):  
  return abs(n - 50)  
  
thislist = [100, 50, 65, 82, 23]  
thislist.sort(key = myfunc)  
print(thislist)

The function will return a number that will be used to sort the list (the lowest number first):

## Copy a List

You cannot copy a list simply by typing list2 = list1, because: list2 will only be a reference to list1, and changes made in list1 will automatically also be made in list2.

There are ways to make a copy, one way is to use the built-in List method copy().

### Example

Make a copy of a list with the copy() method:

thislist = ["apple", "banana", "cherry"]  
mylist = thislist.copy()  
print(mylist)

Another way to make a copy is to use the built-in method list().

### Example

Make a copy of a list with the list() method:

thislist = ["apple", "banana", "cherry"]  
mylist = list(thislist)  
print(mylist)

## Join Two Lists

There are several ways to join, or concatenate, two or more lists in Python.

One of the easiest ways are by using the + operator.

### Example

Join two list:

list1 = ["a", "b", "c"]  
list2 = [1, 2, 3]  
  
list3 = list1 + list2  
print(list3)

Another way to join two lists are by appending all the items from list2 into list1, one by one:

### Example

Append list2 into list1:

list1 = ["a", "b" , "c"]  
list2 = [1, 2, 3]  
  
for x in list2:  
  list1.append(x)  
  
print(list1)

Or you can use the extend() method, which purpose is to add elements from one list to another list:

### Example

Use the extend() method to add list2 at the end of list1:

list1 = ["a", "b" , "c"]  
list2 = [1, 2, 3]  
  
list1.extend(list2)  
print(list1)

# Tuple

Tuples are used to store multiple items in a single variable.

Tuple is one of 4 built-in data types in Python used to store collections of data, the other 3 are [List](https://www.w3schools.com/python/python_lists.asp), [Set](https://www.w3schools.com/python/python_sets.asp), and [Dictionary](https://www.w3schools.com/python/python_dictionaries.asp), all with different qualities and usage.

A tuple is a collection which is ordered and **unchangeable**.

Tuple items are ordered, unchangeable, and allow duplicate values.

Tuples are written with round brackets.

### Example

Create a Tuple:

thistuple = ("apple", "banana", "cherry")  
print(thistuple)

## Tuple Items

Tuple items are ordered, unchangeable, and allow duplicate values.

Tuple items are indexed, the first item has index [0], the second item has index [1] etc.

## Ordered

When we say that tuples are ordered, it means that the items have a defined order, and that order will not change.

## Unchangeable

Tuples are unchangeable, meaning that we cannot change, add or remove items after the tuple has been created.

## Allow Duplicates

Since tuple are indexed, tuples can have items with the same value:

### Example

Tuples allow duplicate values:

thistuple = ("apple", "banana", "cherry", "apple", "cherry")  
print(thistuple)

## Unpacking a Tuple

fruits = ("apple", "banana", "cherry")  
  
(green, yellow, red) = fruits  
  
print(green)  
print(yellow)  
print(red)

**Note:** The number of variables must match the number of values in the tuple, if not, you must use an asterix to collect the remaining values as a list.

## Using Asterisk\*

If the number of variables is less than the number of values, you can add an \* to the variable name and the values will be assigned to the variable as a list:

### Example

Assign the rest of the values as a list called "red":

fruits = ("apple", "banana", "cherry", "strawberry", "raspberry")  
  
(green, yellow, \*red) = fruits  
  
print(green)  
print(yellow)  
print(red)

fruits = ("apple", "mango", "papaya", "pineapple", "cherry")  
  
(green, \*tropic, red) = fruits  
  
print(green)  
print(tropic)  
print(red)

## Join Two Tuples

To join two or more tuples you can use the + operator:

### Example

Join two tuples:

tuple1 = ("a", "b" , "c")  
tuple2 = (1, 2, 3)  
  
tuple3 = tuple1 + tuple2  
print(tuple3)

## Multiply Tuples

If you want to multiply the content of a tuple a given number of times, you can use the \* operator:

### Example

Multiply the fruits tuple by 2:

fruits = ("apple", "banana", "cherry")  
mytuple = fruits \* 2  
  
print(mytuple)

# Python Sets

Sets are used to store multiple items in a single variable.

A set is a collection which is both unordered and unindexed.

Sets are written with curly brackets.

Set items are unordered, unchangeable, and do not allow duplicate values.

Duplicate values will be ignored:

myset = {"apple", "banana", "cherry"}

To add one item to a set use the add() method.

### Add an item to a set, using the add() method:

thisset = {"apple", "banana", "cherry"}  
  
thisset.add("orange")  
  
print(thisset)

## Add Sets

To add items from another set into the current set, use the update() method.

Add elements from tropical and thisset into newset:

thisset = {"apple", "banana", "cherry"}  
tropical = {"pineapple", "mango", "papaya"}  
  
thisset.update(tropical)  
  
print(thisset)

## Add Any Iterable

The object in the update() method does not have be a set, it can be any iterable object (tuples, lists, dictionaries etc.).

Add elements of a list to at set:

thisset = {"apple", "banana", "cherry"}  
mylist = ["kiwi", "orange"]  
  
thisset.update(mylist)  
  
print(thisset)

## Remove Item

To remove an item in a set, use the remove(), or the discard() method.

### Example

Remove "banana" by using the remove() method:

thisset = {"apple", "banana", "cherry"}  
  
thisset.remove("banana")  
  
print(thisset)

**Note:** If the item to remove does not exist, remove() will raise an error.

### Example

Remove "banana" by using the discard() method:

thisset = {"apple", "banana", "cherry"}  
  
thisset.discard("banana")  
  
print(thisset)

**Note:** If the item to remove does not exist, discard() will **NOT** raise an error.

## Join Two Sets

There are several ways to join two or more sets in Python.

You can use the union() method that returns a new set containing all items from both sets, or the update() method that inserts all the items from one set into another:

### Example

### The union() method returns a new set with all items from both sets:

set1 = {"a", "b" , "c"}  
set2 = {1, 2, 3}  
  
set3 = set1.union(set2)  
print(set3)

### Example

The update() method inserts the items in set2 into set1:

set1 = {"a", "b" , "c"}  
set2 = {1, 2, 3}  
  
set1.update(set2)  
print(set1)

## Keep ONLY the Duplicates

The intersection\_update() method will keep only the items that are present in both sets.

### Example

Keep the items that exist in both set x, and set y:

x = {"apple", "banana", "cherry"}  
y = {"google", "microsoft", "apple"}  
  
x.intersection\_update(y)  
  
print(x)

The intersection() method will return a new set, that only contains the items that are present in both sets.

### Example

Return a set that contains the items that exist in both set x, and set y:

x = {"apple", "banana", "cherry"}  
y = {"google", "microsoft", "apple"}  
  
z = x.intersection(y)  
  
print(z)

Keep All, But NOT the Duplicates

The symmetric\_difference\_update() method will keep only the elements that are NOT present in both sets.

Example

Keep the items that are not present in both sets:

x = {"apple", "banana", "cherry"}  
y = {"google", "microsoft", "apple"}  
  
x.symmetric\_difference\_update(y)  
  
print(x)

The symmetric\_difference() method will return a new set, that contains only the elements that are NOT present in both sets.

Example

Return a set that contains all items from both sets, except items that are present in both:

x = {"apple", "banana", "cherry"}  
y = {"google", "microsoft", "apple"}  
  
z = x.symmetric\_difference(y)  
  
print(z)

# Dictionary

Dictionaries are used to store data values in key:value pairs.

A dictionary is a collection which is ordered\*, changeable and does not allow duplicates.

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}

Get the value of the "model" key:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
x = thisdict["model"]

There is also a method called get() that will give you the same result:

x = thisdict.get("model")

## Get Keys

The keys() method will return a list of all the keys in the dictionary.

x = thisdict.keys()

## Get Values

The values() method will return a list of all the values in the dictionary.

x = thisdict.values()

## Get Items

The items() method will return each item in a dictionary, as tuples in a list.

x = thisdict.items()

## Update Dictionary

The update() method will update the dictionary with the items from the given argument. If the item does not exist, the item will be added.

The argument must be a dictionary, or an iterable object with key:value pairs.

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
thisdict.update({"year": 2020})

## Change Values

You can change the value of a specific item by referring to its key name:

thisdict["year"] = 2018

Removing Items

There are several methods to remove items from a dictionary:

The pop() method removes the item with the specified key name:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
thisdict.pop("model")  
print(thisdict)

The popitem() method removes the last inserted item

thisdict.popitem()

The del keyword removes the item with the specified key name:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}

del thisdict["model"]

The del keyword can also delete the dictionary completely:

del thisdict

The clear() method empties the dictionary:

thisdict.clear()

Print all key names in the dictionary, one by one:

for x in thisdict:  
  print(x)

You can use the keys() method to return the keys of a dictionary:

for x in thisdict.keys():  
  print(x)

Print all values in the dictionary, one by one:

for x in thisdict:  
  print(thisdict[x])

You can also use the values() method to return values of a dictionary:

for x in thisdict.values():  
  print(x)

Loop through both keys and values, by using the items() method:

for x, y in thisdict.items():  
  print(x, y)

Make a copy of a dictionary with the copy() method:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
mydict = thisdict.copy()  
print(mydict)

Another way to make a copy is to use the built-in function dict().

mydict = dict(thisdict)

# Python If ... Else

If statement:

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

If – else statement:

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

If – elif - else statement:

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")

## Short Hand If

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

## Short Hand 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")

One line if else statement, with 3 conditions:

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

This technique is known as **Ternary Operators**, or **Conditional Expressions**.

## 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.

### Example

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

# 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

## 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

## 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)

## 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")

## 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).

Print each fruit in a fruit list:

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

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)

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!")

**Note:** The else block will NOT be executed if the loop is stopped by a break 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

# 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 either can return data as a result or nothing.

## Creating a Function

In Python a function is defined using the def keyword

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

## Calling a Function

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

From a function's perspective:

A parameter is the variable listed inside the parentheses in the function definition.

def my\_function(fname): fname is parameter  
 print(fname);

An argument is the value that is sent to the function when it is called

**my\_function(“Gaurav”) Gaurav is Argument**

## 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")

## Keyword Arguments

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

This way the order of the arguments does not matter.

def my\_function(child3, child2, child1):  
  print("The youngest child is " + child3)  
  
my\_function(child1 = "Emil", child2 = "Tobias", child3 = "Linus")

## Arbitrary Keyword Arguments, \*\*kwargs

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

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

def my\_function(\*\*kid):  
  print("His last name is " + kid["lname"])  
  
my\_function(fname = "Tobias", lname = "Refsnes")

## Default Parameter Value

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

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

## Return Values

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

def my\_function(x):  
  **return 5 \* x**

print(my\_function(3))

## 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

# Python Lambda

A lambda function is a small anonymous function.

A lambda function can take any number of arguments, but can only have one expression.

x = lambda a : a + 10  
print(x(5))

x = lambda a, b : a \* b  
print(x(5, 6))

x = lambda a, b, c : a + b + c  
print(x(5, 6, 2))

## Why Use Lambda Functions?

The power of lambda is better shown when you use them as an anonymous function inside another function.

Say you have a function definition that takes one argument, and that argument will be multiplied with an unknown number:

def myfunc(n):  
  return lambda a : a \* n

mytripler = myfunc(3)  
  
print(mytripler(11))

# Python Classes and 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

## Create Object

Now we can use the class named MyClass to create objects

p1 = MyClass()  
print(p1.x)

## The \_\_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

**Note:** The self parameter is a reference to the current instance of the class, and is used to access variables that belong to the class.

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

p1.myfunc()

**Note:** The \_\_init\_\_() function is called automatically every time the class is being used to create a new object.

## 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()

## Modify Object Properties

You can modify properties on objects like this:

p1.age = 40

## Delete Object Properties and Objects

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

del p1.age

del p1

## 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

# 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

class Person:  
  def \_\_init\_\_(self, fname, lname):  
    self.firstname = fname  
    self.lastname = lname  
  
  def printname(self):  
    print(self.firstname, self.lastname)

## Create a Child Class

class Student(Person):  
  pass

**Note:** Use the pass keyword when you do not want to add any other properties or methods to the class.

Use the Student class to create an object, and then execute the printname method:

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

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

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

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

**Note:** The child's \_\_init\_\_() function **overrides** the inheritance of the parent's \_\_init\_\_() function.

To keep the inheritance of the parent's \_\_init\_\_() function, add a call to the parent's \_\_init\_\_() function:

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

## 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)

## Add Properties

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

## Add Methods

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