**PYTHON**

VARIABLES

Variables are containers for storing data values.

## ALLOWED

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

## NOT ALLOWED

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

## MULTIPLE VALUES

x, y, z = "Orange", "Banana", "Cherry"

x = y = z = "Orange"

## UNPACK A LIST

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

## GLOBAL VARS

x = "awesome"

def myfunc():

  x = "fantastic"

  print("Python is " + x)

myfunc()

print("Python is " + x)

# TYPES

|  |  |
| --- | --- |
| 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 |

print(type(x))

# CASTING

Casting in python is therefore done using constructor functions:

* int() - constructs an integer number from an integer literal, a float literal (by removing all decimals), or a string literal (providing the string represents a whole number)
* float() - constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer)
* str() - constructs a string from a wide variety of data types, including strings, integer literals and float literals

# STRINGS

## STRING LENGTH

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

a = "Hello, World!"  
print(len(a))

* Check for a Word IN or NOT IN a text:

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

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

## STRING SLICING

b = "Hello, World!"  
print(b[:5])

## STRING METHODS

|  |  |
| --- | --- |
| **Method** | **Description** |
| [capitalize()](https://www.w3schools.com/python/ref_string_capitalize.asp) | Converts the first character to upper case |
| [casefold()](https://www.w3schools.com/python/ref_string_casefold.asp) | Converts string into lower case |
| [center()](https://www.w3schools.com/python/ref_string_center.asp) | Returns a centered string |
| [count()](https://www.w3schools.com/python/ref_string_count.asp) | Returns the number of times a specified value occurs in a string |
| [encode()](https://www.w3schools.com/python/ref_string_encode.asp) | Returns an encoded version of the string |
| [endswith()](https://www.w3schools.com/python/ref_string_endswith.asp) | Returns true if the string ends with the specified value |
| [expandtabs()](https://www.w3schools.com/python/ref_string_expandtabs.asp) | Sets the tab size of the string |
| [find()](https://www.w3schools.com/python/ref_string_find.asp) | Searches the string for a specified value and returns the position of where it was found |
| [format()](https://www.w3schools.com/python/ref_string_format.asp) | Formats specified values in a string |
| format\_map() | Formats specified values in a string |
| [index()](https://www.w3schools.com/python/ref_string_index.asp) | Searches the string for a specified value and returns the position of where it was found |
| [isalnum()](https://www.w3schools.com/python/ref_string_isalnum.asp) | Returns True if all characters in the string are alphanumeric |
| [isalpha()](https://www.w3schools.com/python/ref_string_isalpha.asp) | Returns True if all characters in the string are in the alphabet |
| [isascii()](https://www.w3schools.com/python/ref_string_isascii.asp) | Returns True if all characters in the string are ascii characters |
| [isdecimal()](https://www.w3schools.com/python/ref_string_isdecimal.asp) | Returns True if all characters in the string are decimals |
| [isdigit()](https://www.w3schools.com/python/ref_string_isdigit.asp) | Returns True if all characters in the string are digits |
| [isidentifier()](https://www.w3schools.com/python/ref_string_isidentifier.asp) | Returns True if the string is an identifier |
| [islower()](https://www.w3schools.com/python/ref_string_islower.asp) | Returns True if all characters in the string are lower case |
| [isnumeric()](https://www.w3schools.com/python/ref_string_isnumeric.asp) | Returns True if all characters in the string are numeric |
| [isprintable()](https://www.w3schools.com/python/ref_string_isprintable.asp) | Returns True if all characters in the string are printable |
| [isspace()](https://www.w3schools.com/python/ref_string_isspace.asp) | Returns True if all characters in the string are whitespaces |
| [istitle()](https://www.w3schools.com/python/ref_string_istitle.asp) | Returns True if the string follows the rules of a title |
| [isupper()](https://www.w3schools.com/python/ref_string_isupper.asp) | Returns True if all characters in the string are upper case |
| [join()](https://www.w3schools.com/python/ref_string_join.asp) | Converts the elements of an iterable into a string |
| [ljust()](https://www.w3schools.com/python/ref_string_ljust.asp) | Returns a left justified version of the string |
| [lower()](https://www.w3schools.com/python/ref_string_lower.asp) | Converts a string into lower case |
| [lstrip()](https://www.w3schools.com/python/ref_string_lstrip.asp) | Returns a left trim version of the string |
| [maketrans()](https://www.w3schools.com/python/ref_string_maketrans.asp) | Returns a translation table to be used in translations |
| [partition()](https://www.w3schools.com/python/ref_string_partition.asp) | Returns a tuple where the string is parted into three parts |
| [replace()](https://www.w3schools.com/python/ref_string_replace.asp) | Returns a string where a specified value is replaced with a specified value |
| [rfind()](https://www.w3schools.com/python/ref_string_rfind.asp) | Searches the string for a specified value and returns the last position of where it was found |
| [rindex()](https://www.w3schools.com/python/ref_string_rindex.asp) | Searches the string for a specified value and returns the last position of where it was found |
| [rjust()](https://www.w3schools.com/python/ref_string_rjust.asp) | Returns a right justified version of the string |
| [rpartition()](https://www.w3schools.com/python/ref_string_rpartition.asp) | Returns a tuple where the string is parted into three parts |
| [rsplit()](https://www.w3schools.com/python/ref_string_rsplit.asp) | Splits the string at the specified separator, and returns a list |
| [rstrip()](https://www.w3schools.com/python/ref_string_rstrip.asp) | Returns a right trim version of the string |
| [split()](https://www.w3schools.com/python/ref_string_split.asp) | Splits the string at the specified separator, and returns a list |
| [splitlines()](https://www.w3schools.com/python/ref_string_splitlines.asp) | Splits the string at line breaks and returns a list |
| [startswith()](https://www.w3schools.com/python/ref_string_startswith.asp) | Returns true if the string starts with the specified value |
| [strip()](https://www.w3schools.com/python/ref_string_strip.asp) | Returns a trimmed version of the string |
| [swapcase()](https://www.w3schools.com/python/ref_string_swapcase.asp) | Swaps cases, lower case becomes upper case and vice versa |
| [title()](https://www.w3schools.com/python/ref_string_title.asp) | Converts the first character of each word to upper case |
| [translate()](https://www.w3schools.com/python/ref_string_translate.asp) | Returns a translated string |
| [upper()](https://www.w3schools.com/python/ref_string_upper.asp) | Converts a string into upper case |
| [zfill()](https://www.w3schools.com/python/ref_string_zfill.asp) | Fills the string with a specified number of 0 values at the beginning |

## STRING FORMAT()

age = 36

state = "relaxed"

txt = "My name is John, and I am {} and i like to be very {}"

print(txt.format(age, state))

## STRING ESCAPE

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

|  |  |
| --- | --- |
| **Code** | **Result** |
| \' | Single Quote |
| \\ | Backslash |
| \n | New Line |
| \r | Carriage Return |
| \t | Tab |
| \b | Backspace |
| \f | Form Feed |
| \ooo | Octal value |
| \xhh | Hex value |

# BOOLEANS

print(10 > 9) TRUE  
print(10 == 9) FALSE  
print(10 < 9) FALSE

## BOOLEANS – ALL FALSES

print(bool(False))

print(bool(None))

print(bool(0))

print(bool(""))

print(bool(()))

print(bool([]))

print(bool({}))

## **BOOLEANS – OTHER TYPE OF FALSE**

One more value, or object in this case, evaluates to False, and that is if you have an object that is made from a class with a \_\_len\_\_ function that returns 0 or False:

class myclass():  
  def \_\_len\_\_(self):  
    return 0  
  
myobj = myclass()  
print(bool(myobj))

## OPERATORS

|  |  |  |
| --- | --- | --- |
| **Operator** | **Name** | **Example** |
| + | Addition | x + y |
| - | Subtraction | x - y |
| \* | Multiplication | x \* y |
| / | Division | x / y |
| % | Modulus | x % y |
| \*\* | Exponentiation | x \*\* y |
| // | Floor division | x // y |

|  |  |  |
| --- | --- | --- |
| **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 |

# **COLLECTIONS (ARRAYS)**

There are four collection data types in the Python programming language:

* **List** is a collection which is ordered and changeable. Allows duplicate members.
* [**Tuple**](https://www.w3schools.com/python/python_tuples.asp) is a collection which is ordered and unchangeable. Allows duplicate members.
* [**Set**](https://www.w3schools.com/python/python_sets.asp) is a collection which is unordered, unchangeable\*, and unindexed. No duplicate members.
* [**Dictionary**](https://www.w3schools.com/python/python_dictionaries.asp) is a collection which is ordered\*\* and changeable. No duplicate members.

# 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:

mylist = ["apple", "banana", "cherry"]

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.

## INSERT, APPEND, EXTEND, REMOVE DATA

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

thislist.insert(2, "watermelon")

thislist.append("orange")

tropical = ["mango", "pineapple", "papaya"]

thislist.extend(tropical)

thistuple = ("kiwi", "orange")

thislist.extend(thistuple)

thislist.remove("banana")

*# RESULT*

['apple', 'watermelon', 'cherry', 'orange', 'mango', 'pineapple', 'papaya', 'kiwi', 'orange']

## LIST COMPREHENSSION

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

fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

newlist = []

for x in fruits:

  if "a" in x:

    newlist.append(x)

print(newlist)

*# INSTEAD WE CAN DO*

newlist = [x for x in fruits if "a" in x]

print(newlist)

*# OUTPUT*

['apple', 'banana', 'mango']

['apple', 'banana', 'mango']

## LIST METHODS

|  |  |
| --- | --- |
| **Method** | **Description** |
| [append()](https://www.w3schools.com/python/ref_list_append.asp) | Adds an element at the end of the list |
| [clear()](https://www.w3schools.com/python/ref_list_clear.asp) | Removes all the elements from the list |
| [copy()](https://www.w3schools.com/python/ref_list_copy.asp) | Returns a copy of the list |
| [count()](https://www.w3schools.com/python/ref_list_count.asp) | Returns the number of elements with the specified value |
| [extend()](https://www.w3schools.com/python/ref_list_extend.asp) | Add the elements of a list (or any iterable), to the end of the current list |
| [index()](https://www.w3schools.com/python/ref_list_index.asp) | Returns the index of the first element with the specified value |
| [insert()](https://www.w3schools.com/python/ref_list_insert.asp) | Adds an element at the specified position |
| [pop()](https://www.w3schools.com/python/ref_list_pop.asp) | Removes the element at the specified position |
| [remove()](https://www.w3schools.com/python/ref_list_remove.asp) | Removes the item with the specified value |
| [reverse()](https://www.w3schools.com/python/ref_list_reverse.asp) | Reverses the order of the list |
| [sort()](https://www.w3schools.com/python/ref_list_sort.asp) | Sorts the list |

## **UPPERCASE AND LOWERCASE SORTING CAN BE A PROBLEM**

thislist = ["banana", "Orange", "Kiwi", "cherry"]

thislist.sort(key = str.lower) # won´t work

thislist.sort(key = str.lower) # will work

print(thislist)

# TUPLES

mytuple = ("apple", "banana", "cherry")

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**. They allow duplicates since they are indexed.

Tuples are written with round brackets.

CHANGE VALUES

Once a tuple is created, you cannot change its values. Tuples are **unchangeable**, or **immutable** as it also is called.

But there is a workaround. You can convert the tuple into a list, change the list, and convert the list back into a tuple.

## **ADD TUPLE TO A TUPLE**

You are allowed to add tuples to tuples, so if you want to add one item, (or many), create a new tuple with the item(s), and add it to the existing tuple

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

y = ("orange",)

thistuple += y

## REMOVE OR DELETE

y.remove("apple")

The del keyword can delete the tuple completely:

del thistuple  
print(thistuple) #this will raise an error because the tuple no longer exists

## UNPACK

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

## METHODS

|  |  |
| --- | --- |
| **Method** | **Description** |
| [count()](https://www.w3schools.com/python/ref_tuple_count.asp) | Returns the number of times a specified value occurs in a tuple |
| [index()](https://www.w3schools.com/python/ref_tuple_index.asp) | Searches the tuple for a specified value and returns the position of where it was found |

# SETS

myset = {"apple", "banana", "cherry"}

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

A set is a collection which is unordered, unchangeable\*, and unindexed.

**\* Note:** Set items are unchangeable, but you can remove items and add new items.

**Sets cannot have two items with the same value.**

You cannot access items in a set by referring to an index or a key.

thisset = {"apple", "banana", "cherry"}  
  
for x in thisset:  
  print(x)

# DICTIONARIES

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

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

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

## ACCESS ITEMS

x = thisdict["model"]

x = thisdict.get("model")

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

x = thisdict.keys()

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

x = thisdict.values()

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

x = thisdict.items()

## REMOVE ITEMS

thisdict.pop("model")

## LOOP AND COPY DICTIONARIES

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

for x in thisdict:

  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)

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

for x in thisdict.keys():

  print(x)

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

for x in thisdict.items():

  print(x)

*# make a copy*

mydict = thisdict.copy()

print(mydict)

*# Make a copy of a dictionary with the dict() function:*

mydict = dict(thisdict)

## METHODS

|  |  |
| --- | --- |
| **Method** | **Description** |
| [clear()](https://www.w3schools.com/python/ref_dictionary_clear.asp) | Removes all the elements from the dictionary |
| [copy()](https://www.w3schools.com/python/ref_dictionary_copy.asp) | Returns a copy of the dictionary |
| [fromkeys()](https://www.w3schools.com/python/ref_dictionary_fromkeys.asp) | Returns a dictionary with the specified keys and value |
| [get()](https://www.w3schools.com/python/ref_dictionary_get.asp) | Returns the value of the specified key |
| [items()](https://www.w3schools.com/python/ref_dictionary_items.asp) | Returns a list containing a tuple for each key value pair |
| [keys()](https://www.w3schools.com/python/ref_dictionary_keys.asp) | Returns a list containing the dictionary's keys |
| [pop()](https://www.w3schools.com/python/ref_dictionary_pop.asp) | Removes the element with the specified key |
| [popitem()](https://www.w3schools.com/python/ref_dictionary_popitem.asp) | Removes the last inserted key-value pair |
| [setdefault()](https://www.w3schools.com/python/ref_dictionary_setdefault.asp) | Returns the value of the specified key. If the key does not exist: insert the key, with the specified value |
| [update()](https://www.w3schools.com/python/ref_dictionary_update.asp) | Updates the dictionary with the specified key-value pairs |
| [values()](https://www.w3schools.com/python/ref_dictionary_values.asp) | Returns a list of all the values in the dictionary |

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.

## ELIF

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

a = 33

b = 33

if b > a:

  print("b is greater than a")

elif a == b:

  print("a and b are equal")

## SHORT HAND IF

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

## SHORT HAND IF … ELSE

print("A") if a > b else print("B")

## SHORT HAND WITH 3 CONDITIONS

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

## AND | OR | NOT

a = 200

b = 33

c = 500

if a > b and c > a:

  print("Both conditions are True")

# USING OR

if a > b or a > c:

  print("At least one of the conditions is True")

# USING “NOT”

if not a > b:

  print("a is NOT greater than b")

## PASS

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.

if b > a:  
  pass

# LOOPS

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

*# while loop*

i = 0

while i < 6:

  print(i)

  i += 1

  if i == 5:

    break

  if i == 3:

    continue

*# for loop*

fruits = ["apple", "banana", "cherry"]

for x in fruits:

  print(x)

  if x == "banana":

    break

## USING RANGE

The range() function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter: range(2, 6), which means values from 2 to 6 (but not including 6):

for x in range(6):  
  print(x)

for x in range(2, 6):  
  print(x)

# this one starts in 0, skips 3, ends in 9

for x in range(0, 10, 3):

  print(x)

# output => 0,3,6,9

Break the loop when x is 3, and see what happens with the else block:

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

## NESTED LOOPS

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

# FUNCTIONS

In Python a function is defined using the def keyword:

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

## PARAMS OR ARGS?

From a function's perspective:

* A parameter is the variable listed inside the parentheses in the function definition.
* An argument is the value that is sent to the function when it is called.

## 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.

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, \*\*kargs

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.

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

## DEFAULT PARAM VALUE

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

my\_function("Sweden")

my\_function()

## 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.

*# recursion*

*#* [*https://www.youtube.com/watch?v=RXGyeWy1Mvw*](https://www.youtube.com/watch?v=RXGyeWy1Mvw) *(this is a superb example)*

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)

def show\_list(list):

    if len(list) != 0:

       show\_list(list[1:])

       print(list)

show\_list([1,2,3,4])

*# output*

[4]

[3, 4]

[2, 3, 4]

[1, 2, 3, 4]

ANOTHER WAY TO DO IT

def show\_list2(list, i=0):

    if i < len(list):

       show\_list2(list, i+1)

       print(list[i:])

show\_list2([1,2,3,4])

*# output*

*# [4]*

*# [3, 4]*

*# [2, 3, 4]*

*# [1, 2, 3, 4]*

# LAMBDA

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

mydoubler = myfunc(2)

print(mydoubler(11))

*# Or, use the same function definition to make TWO functions a DOUBLER & TRIPLER, in the same program:*

def myfunc(n):

  return lambda a : a \* n

mydoubler = myfunc(2)

mytripler = myfunc(3)

print(mydoubler(11))

print(mytripler(11))

## 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.

class MyClass:  
  x = 5

p1 = MyClass()  
print(p1.x)

output => 5

class Person:

*# 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:*

    def \_\_init\_\_(self, name, age):

        self.name = name

        self.age = age

*# The string representation of an object WITH the \_\_str\_\_() function:*

    def \_\_str\_\_(self):

        return f"{self.name}({self.age})"

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

    def myfunc(self):

        print(f"Hello my name is {self.name} and i'm {self.age}")

p1 = Person("John", 36)

print(p1)

p1.myfunc()

*# delete properties*

del p1.age

*# delete objects*

del p1

*# declare an empty class*

class some:

    pass

## INHERITANCE

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

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

PERSON is the parent

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

STUDENT is the child

class Student(Person):  
  pass

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)

# SCOPE

GLOBAL

A variable created in the main body of the Python code is a global variable and belongs to the global scope.

x = 300  
  
def myfunc():  
  print(x)  
  
myfunc()  
  
print(x)

## GLOBAL KEYWORD

If you need to create a global variable, but are stuck in the local scope, you can use the global keyword.

def myfunc():  
  global x  
  x = 300  
  
myfunc()  
  
print(x)

## LOCAL SCOPE

A variable created inside a function belongs to the local scope of that function, and can only be used inside that function.

## FUNCTION INSIDE FUNCTION

As explained in the example above, the variable x is not available outside the function, but it is available for any function inside the function:

def myfunc():  
  x = 300  
  def myinnerfunc():  
    print(x)  
  myinnerfunc()  
  
myfunc()

## MODULES

Consider a module to be the same as a code library.

A file containing a set of functions you want to include in your application.

To create a module just save the code you want in a file with the file extension .py:

Mymodule.py

def greeting(name):  
  print("Hello, " + name)

person1 = {  
  "name": "John",  
  "age": 36,  
  "country": "Norway"  
}

Import the module named mymodule, and call the greeting function:

import mymodule  
  
mymodule.greeting("Jonathan")

a = mymodule.person1["age"]  
print(a)

ALSO MODULES CAN BE NAMED

import mymodule as mx  
  
a = mx.person1["age"]  
print(a)

AND WE CAN IMPORT ONE THING FROM A MODULE

Import only the person1 dictionary from the module:

from mymodule import person1  
  
print (person1["age"])

## FUNCTION dir()

There is a built-in function to list all the function names (or variable names) in a module. The dir() function:

import platform  
  
x = dir(platform)  
print(x)

## DATES

## MATH

## JSON

*# Convert from JSON to Python:*

*# some JSON:*

x =  '{ "name":"John", "age":30, "city":"New York"}'

*# parse x:*

y = json.loads(x)

*# the result is a Python dictionary:*

print(y["age"])

*# Convert from Python to JSON:*

*# a Python object (dict):*

x = {

  "name": "John",

  "age": 30,

  "city": "New York"

}

*# convert into JSON:*

y = json.dumps(x)

*# the result is a JSON string:*

print(y)

When you convert from Python to JSON, Python objects are converted into the JSON (JavaScript) equivalent:

|  |  |
| --- | --- |
| **Python** | **JSON** |
| dict | Object |
| list | Array |
| tuple | Array |
| str | String |
| int | Number |
| float | Number |
| True | true |
| False | false |
| None | null |

## REGEX

When you have imported the re module, you can start using regular expressions:

Search the string to see if it starts with "The" and ends with "Spain":

import re  
  
txt = "The rain in Spain"  
x = re.search("^The.\*Spain$", txt)

The re module offers a set of functions that allows us to search a string for a match:

|  |  |
| --- | --- |
| **Function** | **Description** |
| [findall](https://www.w3schools.com/python/python_regex.asp#findall) | Returns a list containing all matches |
| [search](https://www.w3schools.com/python/python_regex.asp#search) | [Returns a Match object if there is a match anywhere in the string](https://www.w3schools.com/python/python_regex.asp#matchobject) |
| [split](https://www.w3schools.com/python/python_regex.asp#split) | Returns a list where the string has been split at each match |
| [sub](https://www.w3schools.com/python/python_regex.asp#sub) | Replaces one or many matches with a string |

## PIP

PIP is a package manager for Python packages, or modules if you like.

## TRY…EXCEPT

The try block lets you test a block of code for errors.

The except block lets you handle the error.

The else block lets you execute code when there is no error.

The finally block lets you execute code, regardless of the result of the try- and except blocks.

try:

  print(x)

except:

  print("An exception occurred")

try:

  print(x)

except NameError:

  print("Variable x is not defined")

except:

  print("Something else went wrong")

try:

  print("Hello")

except:

  print("Something went wrong")

else:

  print("Nothing went wrong")

*#   output*

Hello

Nothing went wrong

## USER INPUT

username = input("Enter username:")

print("Username is: " + username)

## STRING FORMATTING

To make sure a string will display as expected, we can format the result with the format() method.

price = 49  
txt = "The price is {} dollars"  
print(txt.format(price))

The format() method allows you to format selected parts of a string.

Sometimes there are parts of a text that you do not control, maybe they come from a database, or user input?

To control such values, add placeholders (curly brackets {}) in the text, and run the values through the format() method:

Format the price to be displayed as a number with two decimals:

txt = "The price is {:.2f} dollars"

If you want to use more values, just add more values to the format() method:

print(txt.format(price, itemno, count))

ALSO we can do indexes and named indexes

quantity = 3  
itemno = 567  
price = 49  
myorder = "I want {0} pieces of item number {1} for {2:.2f} dollars."  
print(myorder.format(quantity, itemno, price))

myorder = "I have a {carname}, it is a {model}."  
print(myorder.format(carname = "Ford", model = "Mustang"))

## FILE HANDLING

File handling is an important part of any web application.

Python has several functions for creating, reading, updating, and deleting files.

The key function for working with files in Python is the open() function.

The open() function takes two parameters; filename, and mode.

There are four different methods (modes) for opening a file:

"r" - Read - Default value. Opens a file for reading, error if the file does not exist

"a" - Append - Opens a file for appending, creates the file if it does not exist

"w" - Write - Opens a file for writing, creates the file if it does not exist

"x" - Create - Creates the specified file, returns an error if the file exists

In addition you can specify if the file should be handled as binary or text mode

"t" - Text - Default value. Text mode

"b" - Binary - Binary mode (e.g. images)

The open() function returns a file object, which has a read() method for reading the content of the file:

f = open("demofile.txt")

IS THE SAME AS:

f = open("demofile.txt", "rt")

f = open("demofile.txt", "r")  
print(f.read())

If the file is located in a different location, you will have to specify the file path, like this:

f = open("D:\\myfiles\welcome.txt", "r")  
print(f.read())

Return the 5 first characters of the file:

print(f.read(**5**))

To write to an existing file, you must add a parameter to the open() function:

"a" - Append - will append to the end of the file

"w" - Write - will overwrite any existing content

Open the file "demofile2.txt" and append content to the file:

f = open("demofile2.txt", "a")  
f.write("Now the file has more content!")  
f.close()  
  
#open and read the file after the appending:  
f = open("demofile2.txt", "r")  
print(f.read())

Open the file "demofile3.txt" and overwrite the content:

f = open("demofile3.txt", "w")  
f.write("Woops! I have deleted the content!")  
f.close()  
  
#open and read the file after the overwriting:  
f = open("demofile3.txt", "r")  
print(f.read())

To create a new file in Python, use the open() method, with one of the following parameters:

"x" - Create - will create a file, returns an error if the file exist

"a" - Append - will create a file if the specified file does not exist

"w" - Write - will create a file if the specified file does not exist

f = open("myfile.txt", "x")

Create a new file if it does not exist:

f = open("myfile.txt", "w")

To delete a file, you must import the OS module, and run its os.remove() function:

import os  
os.remove("demofile.txt")

Check if file exists, *then* delete it:

import os  
if os.path.exists("demofile.txt"):  
  os.remove("demofile.txt")  
else:  
  print("The file does not exist")

Remove the folder "myfolder":

import os  
os.rmdir("myfolder")