QUICK TIPS:

* Once a data/ list/ array/ tuple or dictionary is created or imported in python its stored in the computers RAM, ‘address’ is the place where data is stored

print (id(data) # gives the uniqye address code where it is stored

* Python is case sensitive be careful with upper and lower case
* No special character is allowed except “\_”
* Comments in python starts with “#”
* Multi line comments are written between triple quotes [“”” This is a comment “””]
* Python keywords cannot be used for naming any variable or create new function

import keywords

print (keywords.kwlist) # all available keywords

print(len(kewords.kwlist) # number of keywords reserved in python

# ‘len()’ is used to count the nunber of keywords in kwlist, its full form 🡪 lenght

VARIABLES IN PYTHON

Variables are containers for storing data values. Unlike other programming languages, Python has no command for declaring a variable. A variable is created the moment you first assign a value to it. String variables can be declared either by using single or double quotes. 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)

x = 5 # assigning a numerical value to the variable x 🡪 is of type int [integer]

y = “Jerome” # assigning a string value to the variable y 🡪 is of type str [string]

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

print(x)

print(y)

print(z)

x = y = z = "Orange"

print(x)

print(y)

print(z)

DATA TYPES IN PYTHON

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, long, float, complex
* Sequence Types : list, tuple, range
* Mapping Type : dict
* Set Types : set, frozenset
* Boolean Type : bool
* Binary Types : bytes, bytearray, memoryview

You can get the data type of any object by using the **type()** function.

x = 5

print(type(x))

Now we will look into each datatype:

x = “hello World” # str 🡪 string

x = 20 # int 🡪 integer

x = 20.5 OR z = -87.7e100 # float 🡪 numerical numbers with decimal places

x =5j # complex

x = [“apple”, “banana”,”Cherry”] # list 🡪 created with third bracket

x = (“apple”, “banana”,”Cherry”) # tuple 🡪 created with first bracket bracket

x = range(6) # range

x = {"name" : "John", "age" : 36} # dict 🡪 dictionary

x = {"apple", "banana", "cherry"} # set

x = frozenset({"apple", "banana", "cherry"}) # frozenset

x = True # bool 🡪 boolean vales [True, False]

x = b"Hello" # bytes

x = bytearray(5) # bytearray

x = memoryview(bytes(5)) # memoryview

There may be times when you want to specify a type on to a variable. This can be done with casting. Python is an object-orientated language, and as such it uses classes to define data types, including its primitive types. Casting in python is therefore done using constructor functions:

y = int(2.8) # y will be 2

x = float(1) # x will be 1.0

z = str(3.0) # z will be '3.0'

There are four **numeric types** in Python:

* int 🡪 Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length.
* long
* float 🡪 Float, or "floating point number" is a number, positive or negative, containing one or more decimals. Float can also be scientific numbers with an "e" to indicate the power of 10.
* complex 🡪 Complex numbers are written with a "j" as the imaginary part.

**String** literals 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.

df = “ A quick Brown FOX, jumped over, the - fence”

df.split(“,”)

df.rstrip() # removes spaces from right

df.rstrip(-) # removes the value provided

df.lstrip() # removes spaces from left

print(df[0]) # print first character

print(df[-2]) # second last characher

print(df[0:5]) # substring character from 0 to 4

print(df[1 :: 2]) # skip every secong value from index 1

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

|  |  |  |
| --- | --- | --- |
| **FUNCTION** | **DESCRIPTION** | **EXAPMLE** |
| strip() | method removes any whitespace from the beginning or the end | *print(df.strip())* |
| lower() | method returns the string in lower case | *print(df.lower())* |
| upper() | method returns the string in upper case | *print(df.upper())* |
| replace() | method replaces a string with another string | *print(df.replace("H", "J"))* |
| split() | method splits the string into substrings if it finds instances of the separator | *print(df.split(","))* |
| in or not in | To check if a certain phrase or character is present in a string | *print("ain" in df)* |
| + | To concatenate, or combine, two strings you can use the + operator | *df = a + “ “ + b* |
| [capitalize()](https://www.w3schools.com/python/ref_string_capitalize.asp) | Converts the first character to upper case | *df.capitalize()* |
| [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 |  |
| [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) | Joins the elements of an inerrable to the end of the string |  |
| [ljust()](https://www.w3schools.com/python/ref_string_ljust.asp) | Returns a left justified version of the string |  |
| [lstrip()](https://www.w3schools.com/python/ref_string_lstrip.asp) | Returns a left trim version of the string |  |
| maketrans() | 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 |  |
| [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 |  |
| [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 |  |
| [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() | Returns a translated string |  |
| [zfill()](https://www.w3schools.com/python/ref_string_zfill.asp) | Fills the string with a specified number of 0 values at the beginning |  |

We cannot combine a string a variable using “+” concatenation function but we can combine strings and numbers by using the format() method.

age = 36

txt = "My name is John, and I am {}"

print(txt.format(age))

quantity = 3

itemno = 567

price = 49.95

myorder = "I want {} pieces of item {} for {} dollars."

print(myorder.format(quantity, itemno, price))

Escape Character 🡪To insert characters that are illegal in a string, we 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." # this give an error

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

|  |  |
| --- | --- |
| \' | Single Quote |
| \\ | Backslash |
| \n | New Line |
| \r | Carriage Return |
| \t | Tab |
| \b | Backspace |

OPERATORS IN PYTHON

Operators are used to perform operations on variables and values. Python divides the operators in the following groups:

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

Arithmetic operators are used with numeric values to perform common mathematical operations, it follows BEDMAS rule 🡪 Brackets, Exponents, Divisions, Multiplications, Additions & Subtractions:

|  |  |  |  |
| --- | --- | --- | --- |
| Operator | Name | Example |  |
| + | Addition | x + y | 2 + 3 |
| - | Subtraction | x - y |  |
| \* | Multiplication | x \* y |  |
| / | Division | x / y |  |
| % | Modulus | x % y | After division gives the remainder |
| \*\* | Exponentiation | x \*\* y | raised to the power |
| // | Floor division | x // y | After division it removes all decimal points |

x = 10

y = 50

import math # the basic mathematical functions

dir(math) # to see math package functions

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 |

Comparison operators are used to compare two values:

|  |  |  |
| --- | --- | --- |
| Operator | Name | Example |
| == | Equal | x == y |
| != | Not equal | x != y |
| > | Greater than | x > y |
| < | Less than | x < y |
| >= | Greater than or equal to | x >= y |
| <= | Less than or equal to | x <= y |

Logical operators are used to combine conditional statements, performs logical operations and returns True or False as output:

|  |  |  |
| --- | --- | --- |
| perator | Description | Example |
| and | Returns True if both statements are true | x < 5 and x < 10 |
| or | Returns True if one of the statements is true | x < 5 or x < 4 |
| not | Reverse the result, returns False if the result is true | not(x < 5 and x < 10) |

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, test if two variables share an identity:

|  |  |  |
| --- | --- | --- |
| Operator | Description | Example |
| is | Returns true if both variables are the same object | x is y |
| is not | Returns true if both variables are not the same object | x is not y |

x = 10

x is 10

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

|  |  |  |
| --- | --- | --- |
| Operator | Description | Example |
| in | Returns True if a sequence with the specified value is present in the object | x in y |
| not in | Returns True if a sequence with the specified value is not present in the object | x not in y |

pets = [‘dogs’, ‘cats’, ‘parrot’]

‘lion’ in pets

‘dogs’ in pets

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 |

DATA STRUCTURES IN PYTHON

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

* List is a collection which is ordered and changeable. Allows duplicate members.
* Tuple is a collection which is ordered and unchangeable. Allows duplicate members.
* Set is a collection which is unordered and unindexed. No duplicate members.
* Dictionary is a collection which is unordered, changeable and indexed. No duplicate members.

When choosing a collection type, it is useful to understand the properties of that type. Choosing the right type for a particular data set could mean retention of meaning, and, it could mean an increase in efficiency or security.

**LIST** – String and list have similar operations. A list is a general string which can contain of any kind of data type eg. int, string. Every element present in a list will have index assigned to it which starts from 0 to n. List is a collection which is ordered and changeable. In Python lists are written with square brackets “[ ]”. You access the list items by referring to the index number. Remember that the first item has index 0.

df = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]

type(df) # type of data structure df

print(df)

df.count() # Returns the number of elements with the specified value

print(thislist[1]) # Print the second item of the list

df.reverse() # Reverses the order of the list

df.sort() # Sorts the list

df.index() # Returns the index of the first element with the specified value

Negative indexing means beginning from the end, -1 refers to the last item, -2 refers to the second last item etc.

print(df[-1])

print(df[2:5]) # Return the third, fourth, and fifth item

print(df[:4]) # returns the items from the beginning to "orange"

print(df[2:]) # returns the items from "cherry" and to the end

df[1] = "blackcurrant" # Change Item Value

for x in df:

print(x) # Print all items in the list

if "apple" in df:

print("Yes, 'apple' is in the fruits list") # Check if Item Exists in list

print(len(df)) # determine how many items a list has

df.append(“grapes”) # add an item to the end of the list

df.append([“grapes”,”guva”,”lichi”]) # add items to the end of the list

df.insert(1,”watermelon”) # add an item at the specified index

df.remove(”orange”) # removes the specified item

df.pop() # removes the specified index or the last item if index is not specified

df.pop(5) # removes the specified index

del df[5] # del keyword removes the specified index

del df # del keyword can also delete the list completely

df.clear() # clear() method empties the list

df1 = df.copy() # create a copy of df1

id(df) == id(df1) # both data frames are in different memory place

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().

df1 = df.copy()

print(id(df1))

print(id(df))

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

df1 = list(df)

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.

list1 = ["a", "b" , "c"]

list2 = [1, 2, 3]

list3 = list1 + list2

for x in list2:

list1.appen(x)

list1.extend(list2) # add elements from one list to another list

**Tuples** – A tuple once created cannot be modified. First bracket is used to create a tuple.

tup = (‘a’, ‘b’, ‘c’, ‘d’)

tup1 = (‘e’, ‘f’)

tup2 = tup + tup1 # concatenate

tup \* 2 # repetation

tup[2] # indexing

print(tup[1:3])

**Dictionaries** – A group of values within curly bracket

Dict = {'Name' : ['Jerome','Jenny','John'], 'Age' : [30, 26, 20]}

len(Dict)

print(Dict)

Dict.keys()

Dict.values()

print(list(Dict.keys()))

print(list(Dict.values()))

Dict1 = {'Height':[5.8,5.8,5.2]}

Dict.update(Dict1)

print(Dict)

print(len(Dict))

del Dict["Height"]

print(Dict)

**Sets** – Sets are a way to get the unique elements out of a collection. Unordered collection of items within {}. key value not required. Set gives unique values of elements.

St = {1,2,3,4,5,5,5,6,6}

print(St)

St1 = {5,6,'a','b','c'}

St | St1 # | gives the union of two sets St & St1

St & St1 # gives the common elemts from both the sets [intersection]

St - St1

St1 – St

df = ["apple","banana","orange","grapes","orange","apple"]

df1 = set(df)

df2 = set(["kiwi","orange","avacado","peach"])

print(df1)

print(df1.intersection(df2))

print(df1.union(df2))

print(df1.difference(df2))

print(df1.symmetric\_difference(df2))

"kiwi" in df2

LOOPS IN PYTHON

Python flow control defines the flow of execution of program

1. if\_else
2. nested if\_else
3. for
4. while (till the point)
5. break
6. continue

in python we start a loop with “:” semi-colon symbol.

single tab or four spaces 🡪 indentation [The space before a line when it is a continuation of a loop statement

if (contition) :

statement 1

else :

statement 2

a = 70

if a >50 :

print(“what ever you feel like”)

else :

difference = 50 – a

print(“ a is smaller than 50 by” + str(difference) + “unit”)

if (contition1) :

statement 1

elif (condition2)

statement 2

else :

statement 3

n1 = 2

n2 = 60

n3 = 78

if (n1 <= n2) and (n1 <= n3) :

print (“Smallest n1”)

elif (n2 <= n1) and (n2 <= n3) :

print (“Smallest n2”)

else :

print (“smallest n3”)

for iterating\_var in sequence :

execute statement

fruits = [‘apple’, ‘cherry’, ‘banana’]

for x in fruits :

print x

for I in range (3) :

print (len(fruits[i])) # python base fuction - range(start, stop, spet)

My\_string = “Hello Jenny”

for n, alphabet in enumerarte (My\_string) :

print (alphabent, n)

while (condition is True) :

statement 1

a = 1

while a < 3 :

print (a)

a = a + 2

while a < 3 :

if (a % 2 == 0)

print (a, “is even”)

else :

print (a, “is odd”)

a = a +1

a = 10

while a > 0:

a = a -1

if (a != 5) :

print (a)

else :

break

a = 10

while a > 0:

a = a -1

if (a != 5) :

print (a)

else :

continue

Function are of two types : user defines and built in, user defines functions are written as per our convenience and start with ***def***

def function\_name (arg1,arg2,arg3,…….) :

“”” function documentation

line1

line 2 “””

statement

return

print (function\_name.\_\_doc\_\_) # see the functions documentation

def tripple (x) :

return 3 \* x

def cube (x) :

out = x \*\* 3

return out

def factorial (n) :

if n > 1 :

return n \* factorial(n-1)

else :

return (n)

def addition (\*args) : # \* when the range is unklnown

return (sum(args))

**Lamda** is an anonymous function which can take any number of arguments

lamda arguments : expression

x = lamda a: a + 10

print(x(5))

NUMPY

print(np.\_\_version\_\_) # two consecutive underscores, to version of numpy package

df = np.arange(1,50,5) # (start,stop,step)

df.size

np.linspace( ) # create equally spaced arrays

np.ones(7)

np.zero((5,4)) #[Rows, Columns]

PANDAS

WORKING WITH TIME DATA IN PYTHON