Python Tutorial

Wednesday, 14 July 2021 00:23

Python:

Reference:
https://www.w3schools.com/python/python_reference.asp

Check python version on system

```
python --version
python3 --version
```

Indent:

The number of spaces is up to you, but it has to be at least one and should same style through out the indent block.

Comment:

For comment the line we use #
But when we want to comment multiple line then:
line 1
#line 2
#line 3

Also if we use a string and that not define in any variable then it will act as multiple line comment (means program ignore the string written in just on open area)
E.g. '"This line will ignore when run the code"'

'this line also ignore when run the code'

A='this line act as string assign in variable A'

You can use ' or " in string define.

Variables

Any variable define out side of function is **Global variables**
E.g. x = "ram"

We take user input by using **input()**

We take user input by using **input()**

If we want call variable in function then write **x** or **global x**
Inside the function x is "ram"

But if you assign **x= "sita"** inside the function then it act as **local variable** within that particular function.

If you call **global x** inside function and then assign x=**"sita"** then **x** will be overwrite and update **"ram"** by **"sita"**

Variable should be start by _ (underscore) or Character
Variable may contain Alpha-numeric and start only by character or _

thisIsCamelType
ThisIsPascalType
this_is_snake_type

Data Type:

Text Type:	Str
Numeric Types:	int, float, complex
Sequence Types:	list, tuple, range
Mapping Type:	Dictionary
Set Types:	set, frozenset
Boolean Type:	Bool
Binary Types:	bytes, bytearray, memoryview

List ["this", "is", "list"] can be change further operation (Mutable) like a array.
Tuple ("this", "is", "tuple") is fixed (immutable)
Dictionary {"Name" : "Ram", "age" : "24"} is key and value
Set {"this", "is", "set"} has not repeatable elements

List - ordered and changeable. Allows duplicate members. Can mix data type

Tuple - ordered and unchangeable. Allows duplicate members. can mix data type

Set - unordered and unindexed. No duplicate members. Can mix data type

Dictionary - ordered and changeable. No duplicate members.

Casting or constructor:

It construct one data type to other or can say convert data type is called casting.

```
x = int(1)     # x will be 1
y = int(2.8) # y will be 2
z = int("3") # z will be 3

x = str("s1") # x will be 's1'
y = str(2)    # y will be '2'
z = str(3.0)  # z will be '3.0'
```

String:

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

in the result, the line breaks are inserted at the same position as in the code

***String act as a array it means if `print(a[1])` its output is "o"**

***Looping in string:**

E.g.

```
for x in "banana":
  print(x)

Output:
     b
     a
     n
     a
     n
     a
```

*We can check character or word in the string.

E.g.

```
A="this is test string"

print("s" in A)
```

```
Output is: True

print("test" in A)
Output is: True
```

*We can slice string:
E.g.

```
b = "Hello, World!"
print(b[2:])

Output:
llow, World!
```

*Some string operation command

A="string"
A.upper() -convert in upper case
A.lower() -convert in lower case
A.replace() -replace particular character with new one
A.split() -split string at specific position
A.strip() -remove space from start and end of string, not in middle part of string

***Format String -You can format integer as string without change the data type. It just insert the value in the placeholder of string.**

E.g.

```
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.
```

*Also we can use placeholder indexing:

```
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
```

*If you want inverted comma in the string then by default it is not possible so use escap character like this:

E.g.

```
txt = "I want this inverted \"comma\" in the string, then use escap character."
print(txt)
```

```
Output:
 I want this inverted "comma" in the string, then use escap character.
```

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

***String Methodes**

Method	Description
capitalize()	Converts the first character to upper case
casefold()	Converts string into lower case
center()	Returns a centered string
count()	Returns the number of times a specified value occurs in a string
encode()	Returns an encoded version of the string
endswith()	Returns true if the string ends with the specified value

expandtabs()	Sets the tab size of the string
find()	Searches the string for a specified value and returns the position of where it was found
format()	Formats specified values in a string
format_map()	Formats specified values in a string
index()	Searches the string for a specified value and returns the position of where it was found
isalnum()	Returns True if all characters in the string are alphanumeric
isalpha()	Returns True if all characters in the string are in the alphabet
isdecimal()	Returns True if all characters in the string are decimals
isdigit()	Returns True if all characters in the string are digits
isidentifier()	Returns True if the string is an identifier
islower()	Returns True if all characters in the string are lower case
isnumeric()	Returns True if all characters in the string are numeric
isprintable()	Returns True if all characters in the string are printable
isspace()	Returns True if all characters in the string are whitespaces
istitle()	Returns True if the string follows the rules of a title
isupper()	Returns True if all characters in the string are upper case
join()	Joins the elements of an iterable to the end of the string
ljust()	Returns a left justified version of the string
lower()	Converts a string into lower case
lstrip()	Returns a left trim version of the string
maketrans()	Returns a translation table to be used in translations
partition()	Returns a tuple where the string is parted into three parts
replace()	Returns a string where a specified value is replaced with a specified value
rfind()	Searches the string for a specified value and returns the last position of where it was found
rindex()	Searches the string for a specified value and returns the last position of where it was found
rjust()	Returns a right justified version of the string
rpartition()	Returns a tuple where the string is parted into three parts
rsplit()	Splits the string at the specified separator, and returns a list
rstrip()	Returns a right trim version of the string

split()	Splits the string at the specified separator, and returns a list
splitlines()	Splits the string at line breaks and returns a list
startswith()	Returns true if the string starts with the specified value
strip()	Returns a trimmed version of the string
swapcase()	Swaps cases, lower case becomes upper case and vice versa
title()	Converts the first character of each word to upper case
translate()	Returns a translated string
upper()	Converts a string into upper case
zfill()	Fills the string with a specified number of 0 values at the beginning

Booleans:

Any string is True, except empty strings.

Any number is True, except 0.

Any list, tuple, set, and dictionary are True, except empty ones.

Operators:

*Arithmetic 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

*Assignment Operators:

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

*Other Operators

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
is		
Is not		
in		
In not		

Python List

List - ordered and changeable. Allows duplicate members. Can mix data type

Tuple - ordered and unchangeable. Allows duplicate members. can mix data type

Set - unordered and unindexed. No duplicate members. Can mix data type

Dictionary - ordered and changeable. No duplicate members.

*List can be created by []

```
A= [10,20,15,40,25]
```

*List items are ordered, changeable, and allow duplicate values
Ordered means – elements in the list always in fix index, if we add new element in the list then new element go in last index of the list.

*List can contain different data type

```
A= ["abc", 34, True, 40, "male"]
```

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.

Unordered means that the items in a set do not have a defined order.

Set items can appear in a different order every time you refresh, and cannot be referred to by index or key.

Duplicate not allowed means if set have duplicate values then in result it shows only one element

E.g.

```
A ={"a","b","c","a","d"}
```

```
Output:
```

```
{"b","a","c","d"}
```

*In List if you add element on fix index or range [1 : 3] then it overwrite the index value. If you add more than the index element then it act as insert function in List.

.insert()	Insert at specific index
.append()	Insert at last in list
.extend()	Add **B** list in List **A ,** it is same as adding List **A + B**
.remove()	Remove item in List, if duplicate then first item delete first
.pop()	Last item pop out, or you can use index
.del[]	Delete index in list or complete List or complete tuple
.clear()	Clear all element in list

.sort()	By default it sorting list first number then alphabets
.sort(key = str.lower)	We use key value to customized sort
.reverse()	Reverse order sorting
.copy()	Use for copy List **A** in List **B**, same as **B=A**
.count()	It count the no. Of times appear in list or tuple
.index()	To find the index no. Of item
.add()	In Set is you can add element
.update()	insert list item in set, exclude any duplicate items
.discard()	If item not present in set then no error
.union()	will exclude any duplicate items
.intersection_update()	keep only the items that are present in both sets
.intersection()	return a new set, that contains the items that are present in both sets
.symmetric_difference_update()	only the elements that are NOT present in both sets
symmetric_difference()	return a new set, that contains only the elements that are NOT present in both sets

Tuple can add in List by .extend() Methode, if you want operation in tuple then convert in list first

If want to delete all occurrence then use .remove() in loop

*List and Tuple can be unpack
E.g.

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

Output:
apple
banana
cherry
```

*If Unpack size is different then we can use *

```
fruits = ["apple", "mango", "papaya", "pineapple", "cherry"]
[green, *tropic, red] = fruits

print(green)
print(tropic)
print(red)
```

Output:

```
apple
banana
['cherry', 'strawberry', 'raspberry']
```

Python Dictionary:

E.g.

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

*Dictionary use in terms of large data set information
*Dictionary can be nested

E.g.

```
child1 = {
  "name" : "Emil",
  "year" : 2004
}
child2 = {
  "name" : "Tobias",
```

```
  "year" : 2007
}
child3 = {
  "name" : "Linus",
  "year" : 2011
}

myfamily = {
  "child1" : child1,
  "child2" : child2,
  "child3" : child3
}
```

Method	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

Python If-Else:

E.g.

```
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")
```

***we can use short hand type if else:**
E.g.

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

Python Loops:

Python has two primitive loop commands:

1. while loops
2. for loops

*the `break` statement we can stop the loop even if the while condition is true
*the `continue` statement we can stop the current iteration, and continue with the next
*We can use **`else`** statement with **`while / for`** loop
*if loop with no content, put in the `pass` statement to avoid getting an error

Python Functions:

*Function can use arguments; No arguments; multiple arguments; arbitrary arguments (*arg); keyword arguments(**kwarg); or use **List** as arguments

E.g.

```
def my_function(*kids):
  print("The youngest child is " + kids[2])


my_function("Emil", "Tobias", "Linus")
```

```
my_function("Emil", "Tobias", "Linus")
```

E.g.

```
def my_function(child3, child2, child1):
  print("The youngest child is " + child3)

my_function(child1 = "Emil", child2 = "Tobias", child3 = "Linus")
```

E.g.

```
def my_function(**kid):
  print("His last name is " + kid["lname"])

my_function(fname = "Tobias", lname = "Refsnes")
```

***we can use recursion function:**

It is a function which call itself and return value when end the function

Lambada Function:

Use lambda functions when an anonymous function is required for a short period of time.

E.g.

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

mytripler = myfunc(3)

print(mytripler(11))
```

Output:

```
33
```

Python Classes and Objects:

Python is an object oriented programming language

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 we create when we want to use object of the class. It is similar like inbuilt Method.

```
E.g. A=[1,2,3,4]
A.append(5)
```

Here **.append** act as object.
Similarly, in class we create object, example below we call object **y** from class **MyClass**

```
class MyClass:
  x = 5
  y = 7
  z = 9

p1 = MyClass()
print(p1.y)

Output:
7
```

***Actually we use class to creating object**
In real life we use __int__() function

E.g.

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age



p1 = Person("John", 36)
p2 = Person("Ram", 24)
p3 = Person("shyam", 38)

print(p1.name)
print(p2.name, p2.age)
print(p3.age)
```

Output:

```
John

Ram 24

38
```

*Here we use **class** and **__int__()** function to create object each time we make Person data base using class

Python Iterators:

`__iter__()` and `__next__()`

Lists, tuples, dictionaries, and sets are all iterable objects. They are iterable *containers* which you can get an iterator from.
XXXXXXXXXXXXXXXXX

Python Scope:

Scop is nothing but global variable. Means variable only available in defined function or nested function. If Global variable define then it may call inside or outside of the function.

Python Modules:

It is same as python code library.
A file contain set of function which can include or call in code as import library.
To create modules: just save module file by name and extension **.py**

By this we can create our own library

*To list all the function names (or variable names) in a module. The dir()function can use
E.g.
Let our module file name is **MyModule.py** having this piece of code

```
def myfun1():
    print("hello")

X= "variable"
def myfun2():
    print(X)

def myfun3():
```

```
print(12345)
```

Now we **import** the module **MyModule.py** in our code:

```
import MyModule.py

Print dir(MyModule)
```

```
OUTPUT:
All the variables and function
```

Python Datetime:

*import datetime
*datetime.datetime.now()
*To return specific format datetime then **datetime.datetime.now("%B")**

Directive	**Description**	**Example**
%a	Weekday, short version	Wed
%A	Weekday, full version	Wednesday
%w	Weekday as a number 0-6, 0 is Sunday	3
%d	Day of month 01-31	31
%b	Month name, short version	Dec
%B	Month name, full version	December
%m	Month as a number 01-12	12
%y	Year, short version, without century	18
%Y	Year, full version	2018
%H	Hour 00-23	17
%I	Hour 00-12	05
%p	AM/PM	PM
%M	Minute 00-59	41
%S	Second 00-59	08
%f	Microsecond 000000-999999	548513
%z	UTC offset	+0100
%Z	Timezone	CST
%j	Day number of year 001-366	365

%j	Day number of year 001-366	365
%U	Week number of year, Sunday as the first day of week, 00-53	52
%W	Week number of year, Monday as the first day of week, 00-53	52
%c	Local version of date and time	Mon Dec 31 17:41:00 2018
%x	Local version of date	12/31/18
%X	Local version of time	17:41:00
%%	A % character	%
%G	ISO 8601 year	2018
%u	ISO 8601 weekday (1-7)	1
%V	ISO 8601 weeknumber (01-53)	01

Python JSON:

It is a JavaScript format to write text
JSON is inbuilt package of python json
JSON format is similar as a dictionary.

To change JSON to Python use json.load()
E.g.

```
import json
x = '{ "name":"John", "age":30, "city":"New York"}'
y = json.loads(x)
print(y["city"])
```

```
OUTPUT
Ney York
```

*Python format as dictionary
*But JSON format is text. Means inside the {} all characters read as individual character
*In JSON format all the elements are written in JavaScript format. It is combined with indents between {}, [], ()

*We can sort or separate the value. Default value of separator is , :

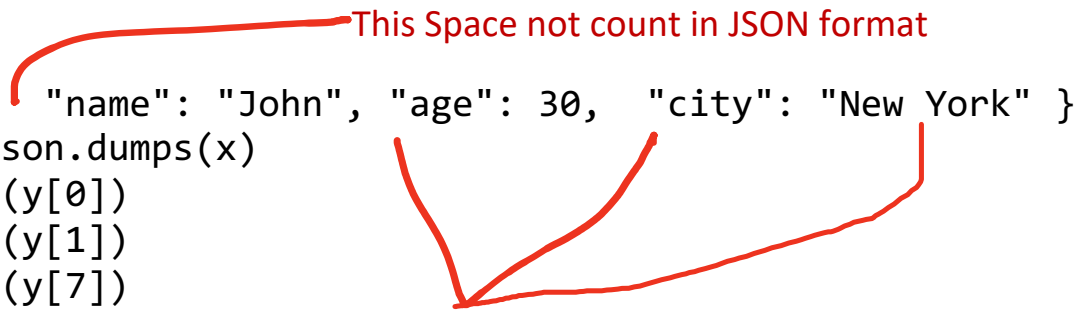
```
*json.dumps(x, indent=4, separators=(". ", " = "))
*json.dumps(x, indent=4, sort_keys=True)
```

To change Python dictionary to JSON use json.dumps()

E.g.

```
import json

x = {  "name": "John", "age": 30,  "city": "New York" }
y = json.dumps(x)
print(y[0])
print(y[1])
print(y[7])
print(y[8])
print(y[9])
print(y[15])
```



These each blank count in JSON format.
Each blank count as 1 space.
Doesn't mater how many No. of spaces in blank

```
OUTPUT
{
"
:
 (it is  blank space)
"
,
```

We can convert Python to JSON

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

Python to JSON format as JavaScript

Python to JSON format as Javascript.

E.g.

import json

```
x =
{"name":"John","age":30,"married":True,"divorced":False,"children
": ("Ann","Billy"), "pets": None, "cars": [{"model": "BMW 230",
"mpg": 27.5}, {"model": "Ford Edge", "mpg": 24.1}]}

print(json.dumps(x, indent=4))
```

OUTPUT

```
{
    "name": "John",
    "age": 30,
    "married": true,
    "divorced": false,
    "children": [
        "Ann",
        "Billy"
    ],
    "pets": null,
    "cars": [
        {
            "model": "BMW 230",
            "mpg": 27.5
        },
        {
            "model": "Ford Edge",
            "mpg": 24.1
        }
    ]
}
```

Python RegEx or Regular Expression:

import re

*It use for **find, find all, split, search, Replace.**

*We can use it for search **end words, start words, first letter, last letter in string etc.**

*We can take help of syntax from
https://www.w3schools.com/python/python_regex.asp

Python PIP:

PIP by default install in python version 3.4 onwards

```
*pip --version can show the version of PIP
*pip install Module
*pip3 install module
*pip uninstall module
*pip3 uninstall module
```

Exception Handling:

There is mainly three handling:
*try
*except
*finally

We can also use **else** if we want bypass except. And particular exception handle like **except NameError:** , **except MathError:**

***finally** block can execute all time with **try** and **except** block

***finally** most use to make program more clean
E.g.

```
try:
  f = open("demofile.txt")
  f.write("Lorum Ipsum")
except:
  print("Something went wrong when writing to the file")
finally:
  f.close()
```

*Raise an exception in program
*we can create our own exception in program, it helps like enter 10 digit mobile no. Or only integer allowed etc.

E.g.

```
x = "hello"
if not type(x) is int:
  raise Exception("Only integers are allowed")
```

Here we can use inbuilt exception error.

E.g.

```
x = "hello"
if not type(x) is int:
  raise TypeError("Only integers are allowed")
```

File Handling:

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

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

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

"w" - Write - Opens a file for writing or it is use for overwrite the file contents, creates the file if it does not exist

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

"t" - Text - Default value. Text mode

"b" - Binary - Binary mode (e.g. images) **`f = open("demofile.txt", 'rb')`**

E.g.

```
f = open("demofile.txt", 'w')
f.write("hello file")
f.close()
```

E.g.

*Loop through the file line by line:

```
f = open("demofile.txt", "r")
for x in f:
  print(x)
```

E.g.

*This code execute first two lines in the file

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

Delete File and Folder:

*Remove the file "demofile.txt":

```
import os
os.remove("demofile.txt")
```

*Remove the folder "myfolder": We can only remove empty folder

```
import os
os.rmdir("myfolder")
```

*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")
```

****** Some Reference for the python******

Color Name and code:
https://www.w3schools.com/colors/colors_names.asp

Matplotlib:
https://www.w3schools.com/python/matplotlib_getting_started.asp