

# PYTHON CHEATSHEET



## 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.

## 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.

A simple line drawing of a pencil, oriented diagonally with the tip pointing towards the bottom right.

## PRINT

print function in Python is a function that outputs to your console window whatever you say you want to print out.

```
print("Hello World!").
```

Output :- Hello, World!

## VARIABLES

Variables are nothing but reserved memory locations to store values.

### Creating Variables

```
Name    = "Jhon"    # type String  
Age      = 30        # type Integer  
Height   = 5.2       # type Float
```





## VARIABLE NAMES

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)
- You can also use the + character to add a variable to another variable.

X = "PYTHON IS "

Y = "AWESOME"

PRINT(X+Y).


OUTPUT: PYTHON IS AWESOME

## PYTHON COMMENTS

- Comments can be used to explain Python code.
- Comments can be used to make the code more readable.
- Comments can be used to prevent execution when testing code.

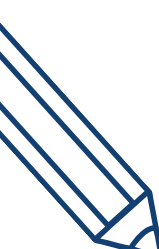
# This is a comment. (For Single Line comment use #)

""" This is also comment.(For Multiple Line comment use """ to the starting point and ending point)



## TYPE CASTING

Type casting is a way to convert a variable from one data type to another data type.

- 
- `int()` - constructs an integer number from an integer literal, a float literal (by rounding down to the previous whole number), 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

## STRING LITERALS

MYSTR = "PYTHON"

PRINT(MYSTR[0]).

OUTPUT: P

MYSTR = "HELLO WORLD"

PRINT(MYSTR[5]).

OUTPUT: (BLANK SPACE).



String literals in python are surrounded by either single quotation marks, or double quotation marks. 'hello' is the same as "hello".

Note :- space is also index value

## STRING SLICES



Hello					
0	1	2	3	4	
-5	-4	-3	-2	-1	

- `s[1:4]` is 'ell' -- chars starting at index 1 and extending up to but not including index 4
- `s[1:]` is 'ello' -- omitting either index defaults to the start or end of the string
- `s[:]` is 'Hello' -- omitting both always gives us a copy of the whole thing (this is the pythonic way to copy a sequence like a string or list)
- `s[1:100]` is 'ello' -- an index that is too big is truncated down to the string length

*str = 'Hello World!'*

*print(str)*

*print(str[0])*

*print(str[2:5])*

*print(str[2:] )*

*print(str + "TEST" )*

*OutPut :-*

*#Hello World*

*#H*

*#llo*

*#llo World*

*#Hello World TEST*





## DATA STRUCTURES

- List

A list is a collection which is ordered and changeable. In Python lists are written with square brackets.

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

Access Items:

```
print(mylist[1])
```

Change Item Value

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

```
mylist[1] = "blackcurrant"
```

```
print(mylist)
```

Add Items

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

```
mylist.append("orange")
```

```
print(mylist)
```

Remove Item

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

```
del mylist[0]
```

```
print(mylist)
```



## DATA STRUCTURES

- Tuple

A tuple is a collection which is ordered and unchangeable. In Python tuples are written with round brackets.

A simple line drawing of a pencil, oriented diagonally with the tip pointing towards the bottom right.

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

Access Items:

```
print(mytuple[1])
```

- Set

A set is a collection which is unordered, unindexed and has no duplicate elements. In Python sets are written with curly brackets.

```
myset = {"apple", "banana", "cherry"}
```

You cannot access items in a set by referring to an index, since sets are unordered the items has no index. But you can loop through the set items using a for loop.

A set with strings, integers and boolean values:

```
set1 = {"abc", 34, True, 40, "male"}
```

```
print(set1)
```

A simple line drawing of an open book, showing its pages and spine, located in the bottom right corner.



## DATA STRUCTURES

- Dictionaries

A tuple is a collection which is ordered and unchangeable. In Python tuples are written with round brackets.

```
mydictionary = {  
    "name":"John", "contact":587456215, "email":"john@example.com"  
}
```

### Change Values

```
mydictionary = {"name":"John", "contact":"587456215", "year":2018}  
mydictionary["year"] = 2019
```

### Add Value

```
mydictionary = {"name":"John", "contact":"587456215", "year":2018}  
mydictionary["email"] = "John@example.com"  
print(mydictionary)
```

### Remove Item

```
mydictionary = {"name":"John", "contact":"587456215", "year":2018}  
del mydictionary ["name"]  
print(mydictionary)
```



## IF ... ELSE

Python supports the usual logical conditions from mathematics:

- Equals:  $a == b$
- Less than:  $a < b$
- Greater than:  $a > b$
- Not Equals:  $a != b$
- Less than or equal to:  $a \leq b$
- Greater than or equal to:  $a \geq b$

## IF STATEMENT:

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

## INDENTATION :

Python relies on indentation, using whitespace, to define scope in the code. Other programming languages often use curly-brackets for this purpose.


If statement, without indentation (will raise an error):

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



## 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")
```

In this example a is equal to b, so the first condition is not true, but the elif condition is true, so we print to screen that "a and b are equal".

## ELSE :

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


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



## LOOPS

Python has two primitive loop commands:

### 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
```

Note: remember to increment i, or else the loop will continue forever

### FOR LOOP:

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

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

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

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



## FUNCTIONS :



- A function is a block of code which only runs when it is called.
- You can pass data, known as parameters, into a function.
- A function can return data as a result.



## CREATING A FUNCTION :

In Python a function is defined using the def keyword:

```
def my_function():  
    print("Hello from a function")
```

## CALLING A FUNCTION :

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

```
def my_function():  
    print("Hello from a function")  
my_function()
```

## PARAMETERS:

Information can be passed to functions as parameter.


Parameters are specified after the function name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma.

```
def my_function(fname):  
    print(fname + " Refsnes")  
my_function("Emil")
```



## RETURN VALUES:

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



```
def my_function(x):  
    return 5 * x  
print(my_function(3))
```

## RECURSION :

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

```
def factorial(n):  
    if n == 1:  
        print(n)  
        return 1  
    else:  
        print (n, '*', end=' ')  
        return n * factorial(n-1)
```

## 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))
```



## CLASSES/OBJECTS :

Python is an object oriented programming language. Almost everything in Python is an object, with its properties and methods. A Class is like a "blueprint" for creating objects.



### CREATE OBJECT :

```
class MyClass:  
    x = 5
```

### CREATE OBJECT :

Create an object named p1, and print the value of x:

```
p1 = MyClass()  
print(p1.x)
```

Multiple Object Example:

```
class myclass():  
    x = 5  
    y = 6  
s1 = myclass()  
print(s1.x)  
s2 = myclass()  
print(s2.x)  
s3 = myclass()  
print(s3.y)
```





## THE `__init__()` FUNCTION

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

To understand the meaning of classes we have to understand the built-in `__init__()` function. All classes have a function called `__init__()`, which is always executed when the class is being initiated. Use the `__init__()` function to assign values to object properties, or other operations that are necessary to do when the object is being created:


```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
p1 = Person("John", 36)
print(p1.name)
print(p1.age)
```

Note: The `__init__()` function is called automatically every time the class is being used to create a new object.

```
class Employee:
    count = 0;
    def __init__(self):
        Employee.count = Employee.count+1
emp = Employee()
emp2 = Employee()
```

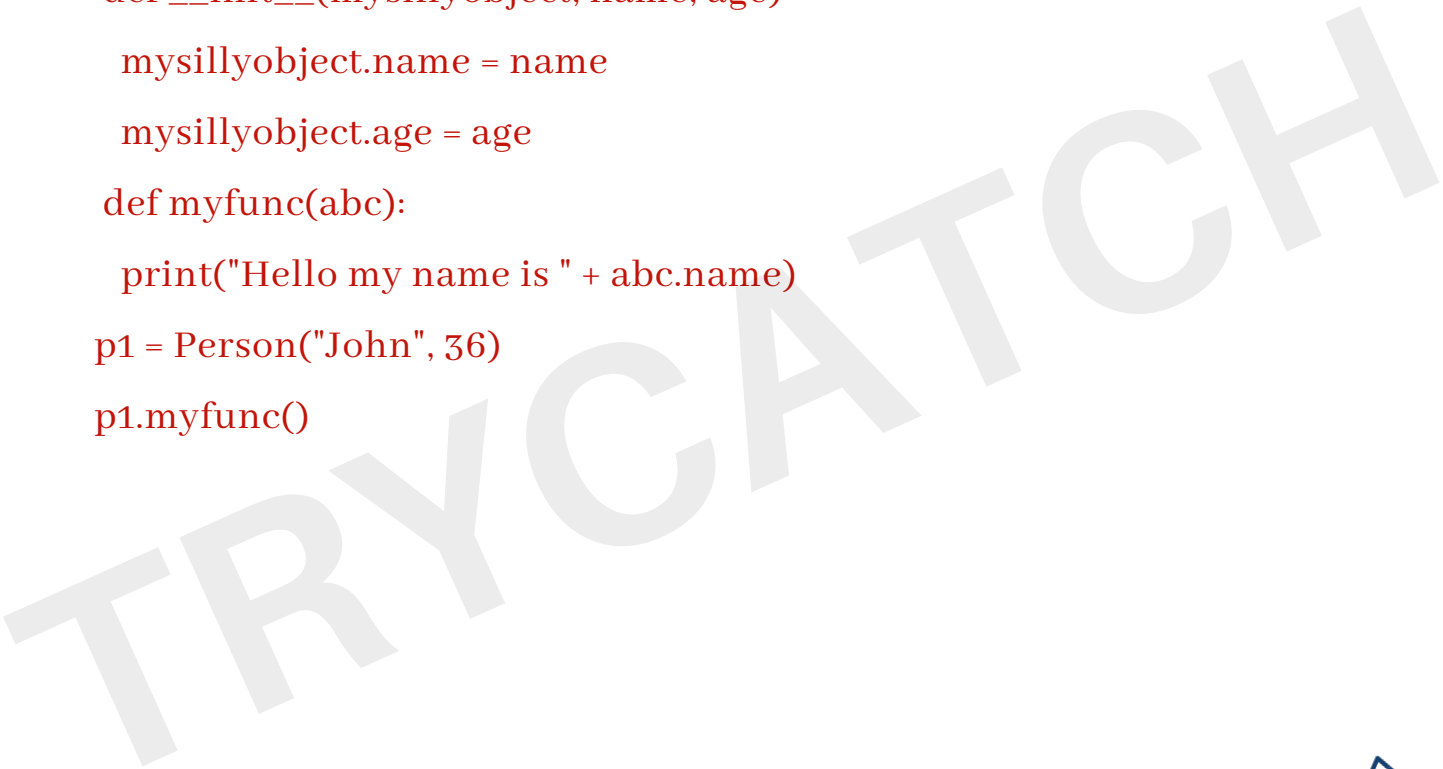


## THE SELF PARAMETER :

A simple line drawing of a pencil, pointing towards the text.

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()
```

A large, light gray, diagonal watermark text reading "TRY CATCH CLASSES" across the center of the page.



{ TRY CATCH }  
CLASSES

THANKYOU