Python

```python
print("hello world ")
```

Indentation

Indentation refers to the spaces at the beginning of a code line.Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.

Comments

Comments starts with a `#`, and Python will ignore them:

```python
#This is a comment
print("Hello, World!")
```

Multiline Comments

you can add a multiline string (triple quotes) in your code, and place your comment inside it:

```python
"""
This is a comment
written in
more than just one line
"""
print("Hello, World!")
```

Variables

Creating Variables

Python has no command for declaring a variable.A variable is created the moment you first assign a value to it.

```python
x = 4       # x is of type int
x = "Sally" # x is now of type str
print(x)
```

Casting

Specify the data type of a variable, this can be done with casting.

```python
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

Get the data type of a variable with the `type()` function.

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

String variables can be declared either by using single or double quotes & Variable names are case-sensitive

Variable Names :

- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number & variable names are case-sensitive

```python
# illegal Variable Names
2myvar = "John"
my-var = "John"
my var = "John"
```

Assign Multiple Values :

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

```python
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)
```

Unpack a Collection

If you have a collection of values in a list, tuple etc. Python allows you to extract the values into variables. This is called *unpacking*.

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

Output Variables :

`print()` function, you output multiple variables, separated by a comma,You can also use the `+` operator to output multiple variables.

```python
x = "Python"
y = "is"
z = "awesome"
print(x, y, z)  #Output: Python is awesome
```

```
x = "Python "
y = "is "
z = "awesome"
print(x + y + z)  #Output: Pythonisawesome
```

For numbers, the `+` character works as a mathematical operator:

```python
x = 5
y = 10
print(x + y)  #Output: 15
```

```python
x = 5
y = "John"
print(x, y)  #Output: 5 John (Concatenation)
```

```python
x = 5
y = "John"
print(x + y) #Output: error (since x is a int and y is a string)
```

Global Variables

Variables that are created outside of a function are known as global,Global variables can be used by everyone, both inside of functions and outside.

```python
x = "awesome"

def myfunc():
  print("Python is " + x)

myfunc()
```

If you create a variable with the same name inside a function, this variable will be local, and can only be used inside the function. The global variable with the same name will remain as it was, global and with the original value.

```python
x = "awesome"

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

myfunc()                    #Output: Python is fantatastic
print("Python is " + x)     #Output: Python is awesome
```

global Keyword :

To create a global variable inside a function, you can use the `global` keyword.

```python
def myfunc():
  global x
  x = "fantastic"

myfunc()

print("Python is " + x)
```

Data Types

Built-in Data 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`

Numbers

```python
x = 1    # int
y = 2.8  # float
z = 1j   # complex
```

Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length.

Float, or "floating point number" is a number, positive or negative, containing one or more decimals.

Complex numbers are written with a "j" as the imaginary part:

Random Number:

Python does not have a `random()` function to make a random number, but Python has a built-in module called `random` that can be used to make random numbers.

```python
import random

print(random.randrange(1, 10))
```

Specify a Variable Type :

```python
x = int(1)   # x will be 1
y = int(2.8) # y will be 2
z = int("3") # z will be 3
w = float("4.2") # w will be 4.2
y = float(2.8)   # y will be 2.8
z = str(3.0)  # z will be '3.0'
```

Strings

Strings are Arrays :

Strings in Python are arrays of bytes representing unicode characters.Square brackets can be used to access elements of the string.

```python
a = "Hello, World!"
print(a[1])   #Output: e
```

String Length :

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

To **check** if a certain phrase or character is **present in a string**, we can use the keyword `in`.

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

To **check** if a certain phrase or character is **NOT present in a string**, we can use the keyword `not in`.

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

Slicing :

Return a range of characters by using the slice syntax,Specify the start index and the end index, separated by a colon, to return a part of the string.

```python
#characters from position 2 to position 5 (not included)
b = "Hello, World!"
print(b[2:5])
```

By leaving out the start index, the range will start at the first character:

```python
b = "Hello, World!"
print(b[:5])
```

By leaving out the *end* index, the range will go to the end:

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

Use negative indexes to start the slice from the end of the string

```python
b = "Hello, World!"
print(b[-5:-2])  #Output: orl
```

Upper Case :

The `upper()` method returns the string in upper case

```python
a = "Hello, World!"
print(a.upper())
```

Lower Case :

The `lower()` method returns the string in lower case:

```python
a = "Hello, World!"
print(a.lower())
```

Remove Whitespace :

The `strip()` method removes any whitespace from the beginning or the end:

```python
a = " Hello, World! "
print(a.strip()) # returns "Hello, World!"
```

Replace String :

The `replace()` method replaces a string with another string

```python
a = "Hello, World!"
print(a.replace("H", "J"))
```

Split String :

The `split()` method returns a list where the text between the specified separator becomes the list items.

```python
a = "Hello, World!"
print(a.split(",")) # returns ['Hello', ' World!']
```

String Format:

We can combine strings and numbers by using the `format()` method,The `format()` method takes the passed arguments, formats them, and places them in the string where the placeholders `{}` are:

```python
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))
```

You can use index numbers `{0}` to be sure the arguments are placed in the correct placeholders

```python
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))
```

Escape Character :

An escape character is a backslash `\` followed by the character you want to insert

```python
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

You can evaluate any expression in Python, and get one of two answers, `True` or `False`.

When you compare two values, the expression is evaluated and Python returns the Boolean answer.

```python
x = "Hello"
y = 15
print(bool(x))  #true
print(bool(y))  #true
```

The following will return False:

```python
bool(False)
bool(None)
bool(0)
bool("")
bool(())
bool([])
bool({})
```

Operators :

Operators are used to perform operations on variables and values.

+	Addition	x + y	
-	Subtraction	x - y	
*	Multiplication	x * y	
/	Division	x / y	
%	Modulus	x % y	
**	Exponentiation	x ** y	
//	Floor division	x // y	

Comparison Operators

==	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

Operator	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)	
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	

Bitwise Operators

Operator	Name	Description	Example	
&	AND	Sets each bit to 1 if both bits are 1	x & y	
|	OR	Sets each bit to 1 if one of two bits is 1	x | y	
^	XOR	Sets each bit to 1 if only one of two bits is 1	x ^ y	
~	NOT	Inverts all the bits	~x	
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off	x << 2	
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off	x >> 2	

Arrays

- **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, unchangeable*, and unindexed. No duplicate members.
- **Dictionary** is a collection which is ordered** and changeable. No duplicate members.

Lists

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

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

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, Set, and Dictionary, all with different qualities and usage.

list() Constructor

use the `list()` constructor when creating a new list.

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

```
print(thislist)
```

Access Items

List items are indexed and you can access them by referring to the index number

You can specify a range of indexes by specifying where to start and where to end the range.

```python
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:5])
```

Check if Item Exists

To determine if a specified item is present in a list use the `in` keyword:

```python
thislist = ["apple", "banana", "cherry"]
if "apple" in thislist:
  print("Yes, 'apple' is in the fruits list")
```

Change Item Value

To change the value of a specific item, refer to the index number

```python
thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"
print(thislist)
```

Change a Range of Item Values

```python
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "mango"]
thislist[1:3] = ["blackcurrant", "watermelon"]
print(thislist)
```

Insert Items

The `insert()` method inserts an item at the specified index:

```python
thislist = ["apple", "banana", "cherry"]
thislist.insert(2, "watermelon")
print(thislist)
```

Append Items

To add an item to the end of the list, use the `append()` method:

```python
thislist = ["apple", "banana", "cherry"]
thislist.append("orange")
print(thislist)
# [apple,banana,cherry,orange]
```

Extend List

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

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

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

Remove Specified Item

The `remove()` method removes the specified item.

```python
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)
```

The `pop()` method removes the specified index.

```python
thislist = ["apple", "banana", "cherry"]
thislist.pop(1)
print(thislist)
```

If you do not specify the index, the `pop()` method removes the last item.

The `clear()` method empties the list..The list still remains, but it has no content

Loop Through a List

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

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

thislist = ["apple", "banana", "cherry"]
i = 0
while i < len(thislist):
  print(thislist[i])
  i = i + 1

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

Sort Lists

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

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

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

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

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

So if you want a case-insensitive sort function, use str.lower as a key function:

```python
thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.sort(key = str.lower)
print(thislist)
```

Reverse Order

The `reverse()` method reverses the current sorting order of the elements

```python
thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.reverse()
print(thislist)
```

Copy Lists

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

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

Join Two Lists

```python
list1 = ["a", "b", "c"]
list2 = [1, 2, 3]

list3 = list1 + list2
print(list3)

list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

for x in list2:
  list1.append(x)
```

```
print(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 items are ordered, unchangeable, and allow duplicate values

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

Change Tuple Values

Tuples are **unchangeable**, or **immutable,**But there is a workaround. You can convert the tuple into a list, change the list, and convert the list back into a tuple

```python
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)

print(x)
```

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:

```python
thistuple = ("apple", "banana", "cherry")
y = ("orange",)
thistuple += y

print(thistuple)
```

Remove Items

Tuples are **unchangeable**, so you cannot remove items from it, but you can use the same workaround as we used for changing and adding tuple items:

```python
thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
y.remove("apple")
thistuple = tuple(y)
```

Python, we are also allowed to extract the values back into variables. This is called "unpacking

```python
fruits = ("apple", "banana", "cherry")

(green, yellow, red) = fruits

print(green)
print(yellow)
print(red)
```

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:

```python
fruits = ("apple", "banana", "cherry", "strawberry", "raspberry")

(green, yellow, *red) = fruits

print(green)
print(yellow)
print(red)
```

Loop Through a Tuple

```python
thistuple = ("apple", "banana", "cherry")
for x in thistuple:
  print(x)

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

thistuple = ("apple", "banana", "cherry")
i = 0
while i < len(thistuple):
  print(thistuple[i])
  i = i + 1
```

Sets

Sets are used to store multiple items in a single variable，a set is a collection which is *unordered*, *unchangeable**, and *unindexed*., Duplicates are not allowed.

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

set() Constructor

use the `set()` constructor to make a set.

```python
thisset = set(("apple", "banana", "cherry"))
# note the double round-brackets
print(thisset)
```

Add Items

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

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

```python
thisset = {"apple", "banana", "cherry"}
tropical = {"pineapple", "mango", "papaya"}

thisset.update(tropical)

print(thisset)
```

Remove Item

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

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

If the item to remove does not exist, `remove()` will raise an error. But `discard()` will **NOT** raise any error.

You can also use the `pop()` method to remove an item, but this method will remove a random item, so you cannot be sure what item that gets removed.

The `clear()` method empties the set.

Join Two Sets

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:

```python
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}

set3 = set1.union(set2)
```

```
set1.update(set2)
print(set3)
```

Both `union()` and `update()` will exclude any duplicate items.

`intersection_update()` method will keep only the items that are present in both sets.

```python
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

x.intersection_update(y)
print(x)
```

`symmetric_difference_update()` method will keep only the elements that are NOT present in both sets.

```python
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

x.symmetric_difference_update(y)
print(x)
```

Dictionaries

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.

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

dict() Constructor

Use the `dict()` constructor to make a dictionary.

```python
thisdict = dict(name = "John", age = 36, country = "Norway")
print(thisdict)
```

Get Keys

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

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

```
}
```