https://www.w3schools.com/python/default.asp

alireza10rezaei10@gmail.com

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

Python Indentation(فاصله از سمت چپ)

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.

Python uses indentation to indicate a block of code.

Example

```
if 5 > 2:
  print("Five is greater than two!")
```

Python will give you an error if you skip the indentation:

Example

Syntax Error:

```
if 5 > 2:
print("Five is greater than two!")
```

The number of spaces is up to you as a programmer, but it has to be at least one.

Example

```
if 5 > 2:
 print("Five is greater than two!")
if 5 > 2:
        print("Five is greater than two!")
```

You have to use the same number of spaces in the same block of code, otherwise Python will give you an error:

Example

Syntax Error:

```
if 5 > 2:
 print("Five is greater than two!")
        print("Five is greater than two!")
```

If you try to combine a string and a number, Python will give you an error:

Example

```
x = 5
y = "John"
print(x + y)#give error
```

اینجوری بنویس بجاش:

```
print(x , y)
```

\'	Single Quote	اینارو توی استرینگ ها میتونی بزنی
\n	New Line	

Python Membership Operators

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

Python If ... Else

Python Conditions and If statements

An "if statement" is written by using the `if` keyword.

Example

If statement:

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

Indentation

Python relies on indentation (whitespace at the beginning of a line) to define scope in the code. Other programming languages often use curly-brackets for this purpose.

Example

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
```

Else

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

Example

```
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")
```

And

The `and` keyword is a logical operator, and is used to combine conditional statements:

Example

Test if `a` is greater than `b`, AND if `c` is greater than `a`:

```
a = 200
b = 33
c = 500
```

```
if a > b and c > a:
  print("Both conditions are True")
```

Or

The `or` keyword is a logical operator, and is used to combine conditional statements:

Example

Test if `a` is greater than `b`, OR if `a` is greater than `c`:

```
if a > b or a > c:
  print("At least one of the conditions is True")
```

Nested If

You can have `if` statements inside `if` statements, this is called *nested* `if` statements.

Example

```
x = 41

if x > 10:
  print("Above ten,")
  if x > 20:
    print("and also above 20!")
  else:
    print("but not above 20.")
```

Python While Loops

The while Loop

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

Example

Print i as long as i is less than 6:

```
i = 1
while i < 6:
  print(i)
  i = i + 1
```

The break Statement

With the `break` statement we can stop the loop even if the while condition is true:

Example

Exit the loop when i is 3: (کلا از حلقه میاد بیرون)

```
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

The continue Statement

With the `continue` statement we can stop the current iteration, and continue with the next:

Example

Continue to the next iteration if i is 3:

(اگه اینو بزنی دیگه ادامه حلقه رو نمیره و برمیگرده از اول شرط حلقه رو چک میکنه اگه درست بود حلقه رو شروع میکنه)

```
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)
```

Python Functions

Creating a Function

In Python a function is defined using the `def` keyword:

Example

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

Calling a Function

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

Example

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

my_function()
```

Arguments

Information can be passed into functions as arguments.

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

The following example has a function with one argument (fname). When the function is called, we pass along a first name, which is used inside the function to print the full name:

Example

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

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

چند ورودی هم میتونه داشته باشه مثل:

```
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Emil", "Refsnes")
```

Return Values

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

Example

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

print(my_function(3))
print(my_function(5))
print(my_function(9))
```

Python Variables

In Python, variables are created when you assign a value to it:

Example

Variables in Python:

```
x = 5
y = "Hello, World!"
```

Variables do not need to be declared with any particular *type*, and can even change type after they have been set.

Example

```
x = 4 # x is of type int
x = "Sally" # x is now of type str
```

هر متغیر رو تو یه خط جدا گانه تعریف کن

```
#Illegal variable names: (اسامی غیر مجاز برای متغیر ها)
2myvar = "John"
my-var = "John"
my var = "John"
```

Python Scope

A variable is only available from inside the region it is created. This is called **scope**.

Local Scope

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

Example

A variable created inside a function is available inside that function:

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

myfunc()
```

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:

Example

The local variable can be accessed from a function within the function:

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

myfunc()
```

Global Scope

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

Global variables are available from within any scope, global and local.

Example

A variable created outside of a function is global and can be used by anyone:

```
x = 300

def myfunc():
  print(x)

myfunc()

print(x)
```

Naming Variables

If you operate with the same variable name inside and outside of a function, Python will treat them as two separate variables, one available in the global scope (outside the function) and one available in the local scope (inside the function):

Example

The function will print the local `x`, and then the code will print the global `x`:

```
x = 300

def myfunc():
  x = 200
  print(x)

myfunc()

print(x)
```

```
200
300
```

Global Keyword

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

The `global` keyword makes the variable global.

Example

If you use the `global` keyword, the variable belongs to the global scope:

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

myfunc()

print(x)
```

```
300
```

Also, use the `global` keyword if you want to make a change to a global variable inside a function.

Example

To change the value of a global variable inside a function, refer to the variable by using the `global` keyword:

```
x = 300

def myfunc():
  global x
  x = 200

myfunc()

print(x)
```

```
200
```

Python Lists

Python Collections (Arrays)

List

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

Example

Create a List:

```
My_list = ["apple", "banana", 75,156]
print(my_list)
```

Access Items

You access the list items by referring to the index number:

Example

Print the second item of the list:

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

Negative Indexing

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

Example

Print the last item of the list:

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

Remember that the first item has index 0.

Change Item Value

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

Example

Change the second item:

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

ولی نمیتونیم به ایندکسی که هنوز تعریف نشده مثلا تو

این مثلا به ایندکس 4 یا 5 یا بیشتر مقدار بدیم ... ارور

میده ... باید از متد هایی که پایین میگم استفاده کنیم:

Add Items

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

Example

Using the `append()` method to append an item:

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

Remove Item

Example

The `pop()` method removes the specified index, (or the last item if index is not specified):

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

```
['apple', 'cherry']
```

اگه بهش مقدار ندیم اخری رو پاک میکنه