PYTHON

Python is a high level programming Language developed by guedeo van rossum in 1991.

⇒ It was a successor to the ABC programming Language

features of python

1. easy to learn
2. free and open source (free access)
3. High Level
4. portable - We were written a python code for your windows machine. Now if you want to run it on a MAC you don't need to make changes to the code. same code can run on any machine. This make python portable language.
5. Interpreter - Line by Line
6. Extensible - If needed you can write some of your python code in other Languages like C++.
7. Large Standard Library - python downloads with a large library that you can use, so you don't have to write your own code for every single thing.
8. Dynamically typed - type for a value is decided at run time

print ()

print () function is used to display messages on to the screen

⇒ Whenever we want to write a msg on the screen use single (' ') double (" ") or triple (''') ('''/""") quotes with print fn.

eg :- print ("Hello World")

Syntax :-

```
print (*objects, sep='', end='\n', flush=false,
   file = sys.stdout)
```

Here, *objects is the value to printed

⇒ The sep - seperator is used between the values. It default into a space.

⇒ After all values are printed end is printed, it default into a newline. '\n'

⇒ The file is where the values are printed and its default value is sys.stdout (screen).

Example 1 :-

```
print (10, 2, 3, 'a', sep = '*')
output :-  10* 2* 3* a
```

Example 2 :-

```
print ('python', end=' ')
print ('program')
output :-  python program.
```

⇒ you don't pass any arguments but you still need to put empty parantheses @ the end of print function this will produce an invisible newline character. which inturn will cause a blank line to appear on your screen.

Example.

```
print ('python')
print ()
print ('program')
output :-  python
           - - - - .
           program
```

Quotations in python

⇒ python accept single, double and triple quotes to denote string literals (msg), as long as the same type of quote start and end the string.

⇒ The triple quotes are used to span the string across multiple lines

```
print ("""  I am
            Learning python
            program """)
```

Indentation

⇒ Indentation refers to the space at the begining of code line.

⇒ In other programming langs, indentation is for readability only, indentation in python is very important.

⇒ python use indentation to indicate a block of code.

Example :-

```
If 10 > 0 :
     print ("positive")
```

output :- positive.

⇒ python will give you an Indentation Error if you skip the indentation.

```
If 10 > 0 :
print ("positive")
```

output :- Indentation error

Comments (#)

All characters after the '#' and upto the end of the line are part of the comment and python interpreter ignore them.

eg :-

```
# printing a message
  print ('Hello')
```

use :- to explain code
  to hide the code

Variable

A variable is a named location used to store data in the memory. It is helpful to think variable as a container that hold data and that can change later in the programe.

eg :- a = 10

variable name rules

⇒ variable name should have a combination of letters in lower case (a-z) or upper case (A-Z) or digits (0-9) or an underscore (_).
eg :- Roll_no = 3

⇒ Don't start a variable name with a digit

⇒ Never use special symbols like (@, *, !, % etc.)

⇒ Case Sensitive.

⇒ If you want to create a variable name having two words use underscore to seperate them.

⇒ Create a name that make sense.
eg :- vowel make more sense than 'v'.

⇒ use capital letters possible to declare a constant
eg :- PI = 3.14.

Assigning values to variables

⇒ The assignment statement creates a new variable and gave them values.

⇒ Assignment statement read right to left only

eg:- a=12

Here we assigned a value 12 to the variable 'a'.

changing the value of a variable

```
eg:-  a=3
      print (a)
      a = 1.5
      print (a)
output :- 3
          1.5 .
```

Assigning single values to multiple variables

```
a=b=c=10
print (a)
print (b)
print (c)
output :-
10
10 } output .
10
```

Assigning multiple values to multiple variables

```
a = 10
b = 20
c = 30

a,b,c = 10,20,30 .
print (a)
print (b)        or  print (a,b,c,sep="\n")
print (c)
```

output
```
10
20
30
```

Swap the variables

In other programming langs we use the code like

```
a = 10
b = 20
c = a  # c = 10
a = b  # a = 20
b = c  # b = 10
print (a)
print (b)
```
output :-
```
20
10
```
In python,
syntax
```
var1 , var2 = var2 , var1
```
eg :-
```
a = 10
b = 20
a, b = b, a
print(a)
print (b)
```
output :-
```
20
10 .
```

Constant

⟹ Sometimes you may want to store values in variables. But you don't want to change these values throughout the execution of the program. To do it in other programming langs you can use constant. The constants are like variables, but their values don't change. During the program executed.

⟹ To work around this you can use all capital letters to name a variable to indicate that the variable should be treated as a constant.

⟹ When encountering variables like these you should not change their values. These variables are constant by convention not by rule.

Keywords

Keywords are reserved words in python

⟹ We cannot use a keyword as a variable name, function name or anyother identifier. They are used to define the syntax and structure of the python language.

⟹ keywords are case sensitive

⟹ There are 33+ keywords in python.

⟹ All the keywords except True, False and None are in lower case.

```
import keyword
print (keyword.kwlist)
```

eg:- if, else, None, import etc:-

Identifiers

Identifier is a name given to entities like class, function, variable etc.

Rules for writing Identifiers

⟹ Same as variables

⟹ keywords cannot be used as identifiers.

Literal

Literal is a raw data given to a variable or constant.

Special Literal

Python contains one special literal

None.

⟹ We use it to specify that the field has not been created.

⟹ It represents absence of data

eg:-
```
a=2
if a==None:
    print('not created')
else:
    print(a)
```
output :- 2.

Literal collection

```
l = [1, 2, 3] # List
t = ('a','b',3) # tuple
s = {'a', 10, 'b'} # set
d = {'a':10, 'b':20, 3:30} # dict.
```

Data type

Every value in python has a datatype

⟹ There are various datatypes in python.

I Numeric type
1. Int
2. float
3. Complex

II Sequence type
1. String
2. tuple
3. List

III Set type
1. Set

IV Mapping type
1. dic

V Boolean

True, False.

python string

String is a sequence of characters, written within single or double quotes.

How to Create a String

⇒ String can be created by enclosing characters inside single or double quotes.

⇒ Even triple quotes can be used in python but generally used to represent multi lined string and doc string.

eg:- s = "python program"

How to access characters in a string.

⇒ you can access individual characters using indexing.

⇒ Index number starts from 0 (zero), Indexing operator is `[]`

eg:- print(s[4] ⇒ o
print(s[8] ⇒ r
~~print(s[[illegible]] ⇒ [illegible]~~

⇒ python allow -ve index for its sequence

⇒ The Index of (-1) refers to the last item, -2 to the second last item and so on.

eg:- print(s[-6] ⇒ r.

⇒ Trying to access a character out of Index range will raise an Index Error.

print (s[15]) ⇒ Index Error

⇒ The Index number must be an Integer. We can't float or other types. This will result into Type Error.

print (s[8.2]) ⇒ Type Error.

⇒ We can access range of characters using slicing slicing operator or (:)

eg:- s = "python program"
0 1 2 3 4 5 6 7 8 9 10 11 12 13

print (s[1:6]) ⇒ ython
print (s[-6:-3]) ⇒ rog
print (s[1:12:2]) ⇒ yhnpor
print (s[:5] pytho
print (s[7:] program

Reversing string using slicing

s = "python"
print (s[::-1]
o/p :- nohtyp

String operations

Concatenation

Joining of 2 or more strings into single one called concatenation.

⇒ The + operator does this in python

eg:- s_1 = "Hello"
s_2 = "World" ⇒ Helloworld
print (s_1 + s_2)

⇒ The star operator can be used to repeat the string for a given no. of times.

eg :- S_1 = 'Hello'
print (S_1 * 3)

o/p :- HelloHelloHello.

<u>String membership test</u>

We can test if a substring exist within a string or not using the keyword 'in'.

eg :- S = python
print ("o" in S)
o/p :- True
print ("a" in S)
o/p :- False