

python

Basics



INDEX

■ Introduction

■ Data Types

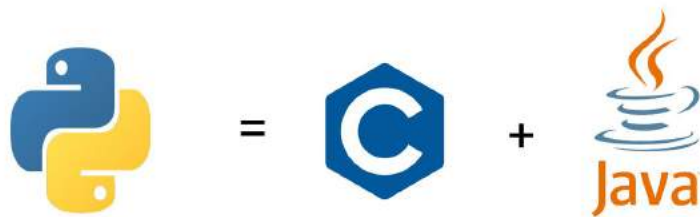
■ Loops/Control Statements



1

Introduction

Python is a programming language that combines the features of C and Java.



It provides elegant style of developing program like C
And also offers class and objects like JAVA

For example, consider below python code:

python program to print Hello world!



History

1. Python laid its foundation in late 1980's.
2. Guido Van Rossum started Implementation of python in December 1989 then later in February 1991 he published the code.(while JAVA developed in 1995)
3. He has developed python language to fill up gap between c and the shell.
4. van Rossum picked the name python from T V show "Monty Python's Flying circus"
5. Logo of python shows two intertwined snakes.
6. Python is open means any body can freely download it and use it to develop programs.

Features

1.Simple and easy to learn:

when we read python we feel like reading english sentences it uses few keywords and structure is simple.

2.Open source and platform independent:

no need to pay for python,it's source code can be read, modified and not dependent on specific OS.

3.High level language and huge library:

high level language use english words to develop program which is easy to learn and read and has a big library which can be used on any OS.

4.Portable and dynamically typed:

when program gives same result on any computer in the world ,then it is called portable and in python no need to declare anything.

5.Database connectivity:-

python provides interfaces to all major databases like ORACLE ,MYSQL,etc....


6.Embedded:

we can insert python programs into C/C+.

Comments

1. Single line comments:

starts with hash symbol(#) and useful to mention that the entire line till the end should be treated as comment.

```
 #program to print hello world  
print("Hello World!")  
  
Hello World!
```

2. Multi line comments:

When we want to make several line as comment into a pair of (""") triple quotes.

```
 '''this program is  
to tell you how triple quotes  
work by printing the hello world'''  
  
print("Hello World!")  
  
Hello World!
```


Indentation

1. A Code of block starts with indentation & ends with first unindicted lines.
- 2.the amount of indentation can be decided by the programmer,but it must be consistent throughout the block.
- 3.Generally 4 white spaces are used.

```
▶ a=10
  b=20
  if a==b:
    print('true')
  else:
    print('false')
```

false

Indentation is given correctly ,
hence no error

```
▶ a=10
  b=20
  if a==b:
    print('true')
  else:
    print('false')
```

File "<ipython-input-6-a82ab6975fb0>", line 4
 print('true')
 ^
IndentationError: expected an indented block

python will give you error if
you skip indentation

```
▶ a=10
  b=20
  if a==b:
    print('true')
  else:
    print('false')
```

File "<ipython-input-7-f80c14e22ade>",
 else:
 ^
IndentationError: unexpected indent

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



Identifiers

“An identifier is a name given to an entity”.

In very simple words, an identifier is a user-defined name to represent the basic building blocks of Python. It can be a variable, a function, a class, a module, or any other object.

Now you know what exactly identifiers are. So, how do we use them? We can't use anything, there are some certain rules to keep in mind that we must follow while naming identifiers.

1. The Python identifier is made with a combination of lowercase or uppercase letters, digits or an underscore.

These are the valid characters.

Lowercase letters (a to z)

Uppercase letters (A to Z)

Digits (0 to 9)

Underscore (_)

Examples of a valid identifier:

num1

FLAG

get_user_name

userDetails

_1234



Variables

1. Variables are containers for storing data value.
2. python has no command to declare variables.
3. a variable is created as you assign a value to it.



```
x=5 #x is variable  
y='python' #y is variable
```

Get the type of the variable using type():



```
x=5 #x is variable  
y='python' #y is variable  
print(type(x))  
print(type(y))
```

```
<class 'int'>  
<class 'str'>
```

Here 'X' is integer and 'Y' is string



Keywords

A python keyword is a reserved word which you can't use as a name of your variable, class, function etc. ...

For example – Python keyword “while” is used for while loop thus you can't name a variable with the name “while” else it may cause compilation error.

There are total 33 keywords in Python 3.6. To display all the keywords

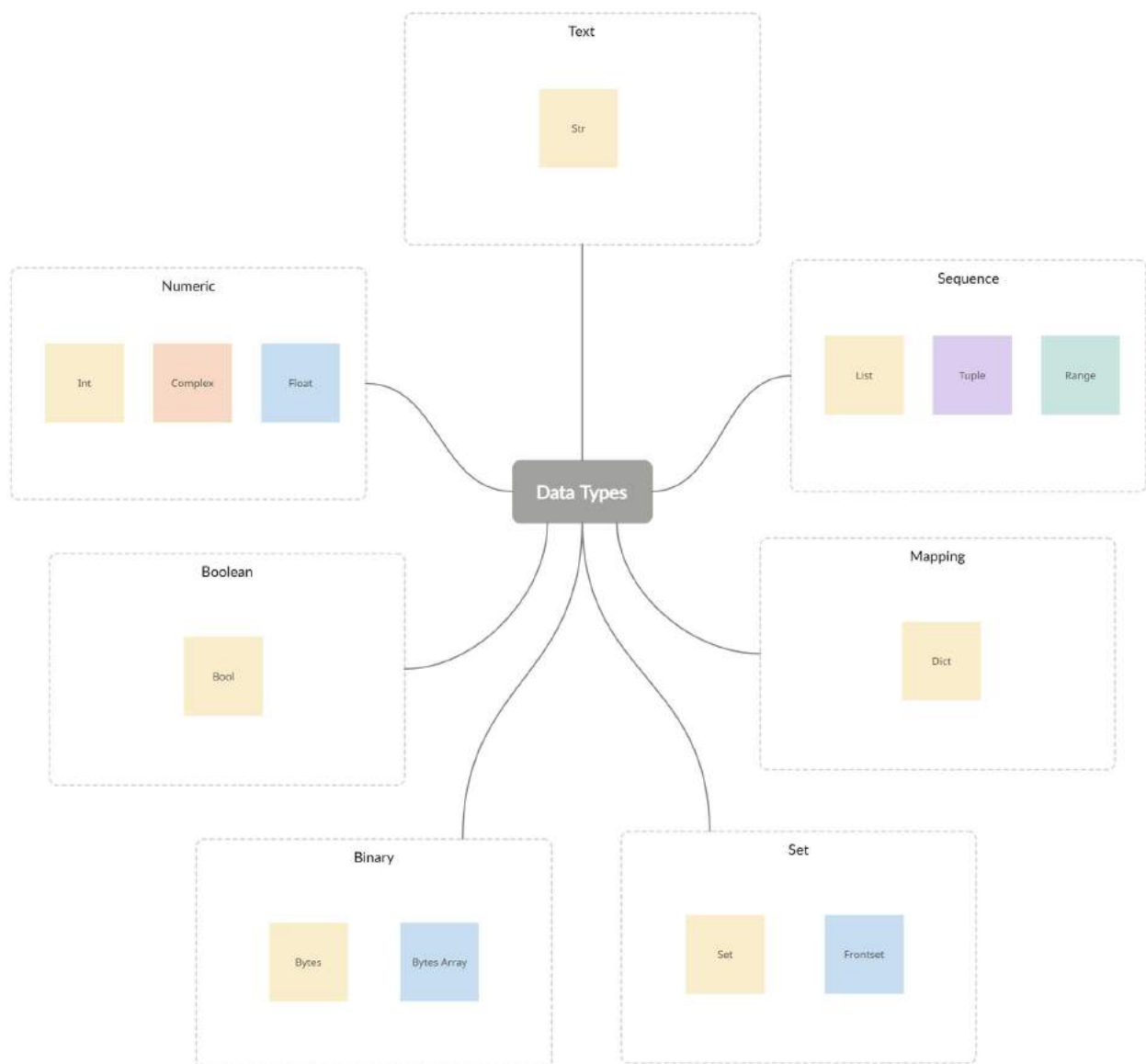


```
import keyword
keyword.kwlist
```


```
['False', 'elif', 'nonlocal', 'None', 'else', 'not', 'True', 'except', 'or', 'and', 'finally', 'pass', 'as', 'for', 'raise', 'assert', 'from', 'return', 'async', 'global', 'try', 'await', 'if', 'while', 'break', 'import', 'with', 'class', 'in', 'yield', 'continue', 'is', 'del', 'def', 'lambda', 'nonlocal']
```

2 Data Types


- 1.Data types represent the type of data stored into a variable .
- 2.Data types already available in python language are called bilt-in data types.
- 3.Data types which are created by programmer are called user-defined data types.




Numeric Datatypes


-  Integer number is a number without a decimal part.
'int' is a class name.

```
 a=10  
print(type(a))  
  
<class 'int'>
```


-  Float datatype represents floating point numbers
they can also written in scientific notations


```
 a=10.0005  
print(type(a))  
  
<class 'float'>
```

-  Complex datatype represents complex numbers
format :a+bj,(j is compulsory at the end of
imaginary part)
a=real value,b=imaginary value


```
 a=10.0005+5j  
print(type(a))  
  
<class 'complex'>
```


Boolean Datatype

-  Bool datatype represents boolean values which can be either true or false.

```
 a= True  
print(type(a))  
  
<class 'bool'>
```

Text Datatype

-  Sequence of characters enclosed within either single/double quotes.(string/str)

```
 a= 'python'  
print(type(a))  
  
<class 'str'>
```

Binary Datatype

bytes datatype represents group of byte numbers just like an array does. A byte number is any positive integer from 0 to 255. content cannot be modified. byte array same as byte but content can be modified.


```
▶ a=[10,20,30,40]
  b=bytes(a)
  print(b[0],b[1],b[2],b[3])
  print(type(b))
```

```
10 20 30 40
<class 'bytes'>
```

```
▶ a=[10,20,30,40]
  b=bytearray(a)
  print(b[0],b[1],b[2],b[3])
  print(type(b))
```

```
10 20 30 40
<class 'bytearray'>
```


Sequence Datatype

 list datatype is an ordered collection of data objects separated by commas and enclosed in square brackets[].

list can contain any type of data such as int,float,str, complex

we can add/update/delete values in the list



```
a=(10,20,30,40)
print(type(a))
```

```
<class 'list'>
```

Range represents sequence of values/numbers.



```
a=range(10)
print(a[0],a[1],a[2],a[3],a[4],a[4],a[5],a[6],a[7],a[8],a[9])
print(type(a))
```

```
0 1 2 3 4 4 5 6 7 8 9
<class 'range'>
```

Range starts with 0 and in the above example we defined range limit to 10 so it takes from 0-9 if we ask to print a[10] it gives error.



```
a=range(10)
print(a[0],a[1],a[2],a[3],a[4],a[4],a[5],a[6],a[7],a[8],a[9],a[10])
print(type(a))
```

```
-----
IndexError                                Traceback (most recent call last)
<ipython-input-36-65511bca5903> in <module>()
      1 a=range(10)
----> 2 print(a[0],a[1],a[2],a[3],a[4],a[4],a[5],a[6],a[7],a[8],a[9],a[10])
      3 print(type(a))

IndexError: range object index out of range
```

Set Datatype

set is a collection which is both unordered & unindexed and enclosed in flower brackets { } and new elements can be added once set is defined.

```
set={1,2,'a','b'}  
print(set)  
set.add(3)  
set.add('z')  
print(set)  
print(type(set))  
  
{1, 2, 'a', 'b'}  
{'z', 1, 2, 3, 'a', 'b'}  
<class 'set'>
```

Frozen set is similar to set only difference is it cannot be modified and it doesn't allow repetitions.

```
[14] a=frozenset('aasghfjah')  
print(a)
```

```
frozenset({'s', 'g', 'a', 'j', 'f', 'h'})
```

```
a.add('rty')
```

```
-----  
AttributeError                                Traceback (most recent call last)  
<ipython-input-15-ed3c1e50076d> in <module>()  
----> 1 a.add('rty')  
  
AttributeError: 'frozenset' object has no attribute 'add'
```

Mapping Datatype

Dictionary represents group of elements arranged in the form of key-value pairs. In the dictionary the first element is considered as a 'key' and the immediate next element is taken as its 'value'.

The key and it's value are seperated by colon(:).

All the key value pairs in dictionary are enclosed in curly brackets { }

we can add or remove values from dictionary

```
dic={'d1':1,'d2':'python'}  
print(dic) #will output all the key value pairs  
print(dic['d1']) #will output only value with 'd1' key '1'  
print(dic.keys()) #will output the list of keys in dic  
print(dic.values()) #will output the list of values in dic  
print(type(dic)) #will output the type of datatype
```

```
{'d1': 1, 'd2': 'python'}  
1  
dict_keys(['d1', 'd2'])  
dict_values([1, 'python'])  
<class 'dict'>
```



3

loops/contional statements

As one of the most basic functions in programming, loops are an important piece to nearly every programming language. Loops enable developers to set certain portions of their code to repeat through a number of loops which are referred to as iterations. This topic covers using multiple types of loops and applications of loops in Python.

Conditional statements, involving keywords such as if, elif, and else, provide Python programs with the ability to perform different actions depending on a boolean condition: True or False. This section covers the use of Python conditionals, boolean logic, and ternary statements.

IF statement

 if statement is used to execute one or more statements depending on whether a condition is true or not. if a condition is true then the statement inside if is executed

Syntax:

If (EXPRESSION == TRUE):

Block of code

else:

Block of code



```
num = 5
if (num < 10):
    print('5 smaller than 10')
```

5 smaller than 10

if-else statements

The statement itself says if a given condition is true then execute the statements present inside the “if block” and if the condition is false then execute the “else” block.

The “else” block will execute only when the condition becomes false. It is the block where you will perform some actions when the condition is not true.

Syntax:

If (EXPRESSION == TRUE):

Statement (Body of the block)

else:

Statement (Body of the block)



```
num = 5
if(num > 10):
    print('number is greater than 10')
else:
    print('number is less than 10')
```

number is less than 10

elif statements

In Python, we have one more conditional statement called “elif” statements.

“elif” statement is used to check multiple conditions only if the given condition is false. It's similar to an “if-else” statement and the only difference is that in “else” we will not check the condition but in “elif” we will check the condition.

Syntax:

if (condition):

 #Set of statement to execute if condition is true

elif (condition):

 #Set of statements to be executed when if condition is false and elif condition is true

else:

 #Set of statement to be executed when both if and elif conditions are false

```
num = 10
if (num == 0):
    print('Number is Zero')

elif (num > 5):
    print('Number is greater than 5')

else:
    print('Number is smaller than 5')
```

Number is greater than 5

While loop

A while loop statement in Python programming language repeatedly executes a target statement as long as a given condition is true. Here, statement(s) may be a single statement or a block of statements. The condition may be any expression, and true is any non-zero value. The loop iterates while the condition is true.

When the condition becomes false, program control passes to the line immediately following the loop.

syntax:

while expression:




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

print( "Good bye!")
```

```
1
2
3
4
5
Good bye!
```


For loop

 It has the ability to iterate over the items of any sequence, such as a list or a string.

If a sequence contains an expression list, it is evaluated first. Then, the first item in the sequence is assigned to the iterating variable `iterating_var`. Next, the statements block is executed. Each item in the list is assigned to `iterating_var`, and the statement(s) block is executed until the entire sequence is exhausted.

Syntax

```
for iterating_var in sequence:  
    statements(s)
```

```
 for letter in 'Python':  
    print (letter)  
  
print ("Good bye!")
```

```
P  
y  
t  
h  
o  
n  
Good bye!
```