Python Programming

by
Dr.T.DHEEPAK
Assistanrt Professor
Department of Computer Science
Government arts and Science College
Perambalur

Rank	Language	Type	Score
1	Python		100.0
2	Java		96.3
3	C		94.4
4	C++		87.5
5	R		81.5
6	JavaScript		79.4
7	C#		74.5
8	Matlab		70.6
9	Swift		69.1
10	Go		68.0

Versions	Releasing Months and Dates
Python 0.9.0	February 1991
Python 1.0	January 1994
Python 2.0	October 2000
Python 2.7.0 - **EOL - Jan 2020**	July 2010
Python 3	December 2008
Python 3.6	December 2016
Python 3.6.5	March 2018
Python 3.7.0	May 2018
Python 3.8	October 2019
Python 3.9	October 2020
Python 3.10-**Current Version**	March 2022

Programming Paradigm

Imperative	Object Oriented	Functional	Procedural
Computation is performed as a direct change to program state. This style is especially useful when manipulating data structures and produces elegant yet simple code. Python fully implements this paradigm.	Relies on data fields that are treated as objects and manipulated only through prescribed methods	Every statement is called as an equation and state or mutable data is avoided. I.e advantage for parallel processing	Tasks are treated as step-by-step iterations where common tasks are placed in functions that are called as needed. This coding style favorsiteration, sequencing, selection, and modularization. Python excels in implementing this particular paradigm

What is Python?

- A Simple, Object Oriented, interpreted, High Level Programming Language.
- It has efficient high level data structures with dynamic typing.
- It is useful for Rapid Application Development as well as for scripting purposes.
- It has extensive library and support modules and packages with code reusability.
- Further mode it is free distributable.

Why Python?

- Increased productivity
- No Compilation process
- Edit - test - debug cycle so fast
- Debugging easy

Conit…

Software Quality : readable code, reuse, maintainable

Developer Productivity: smaller code, easy test debug mechanism

Program portability: Run on any platform

Support libraries : vast amount of packages, third party libraries

Component integration: Invoke C, C++ and Java Libraries, communicate with any framework

Enjoyment: Love to Program

Guido Van Rossum who started a hobby programming project in 1989 during the holidays developed Python Language (named in memory of "Monty Flying Circus"). He inspired from ABC Language and AMOeBa Operating System

- Simple
- Easy to learn
- Free and Open Source
- High level Language

- Interpreted
- Object oriented
- Extensible
- Embedded
- Rapid application development

POWERFUL

- Dynamic Typing
- No variable declaration
- Automatic allocation and garbage collection
- Supports classes, modules and exceptions
- Reusability and structured
- Data Containers

Easy To use

- Type and run
- No compilation and link
- Interactive programming
- Rapid development
- Simple, small and more flexible

statically typed language

A language in which types are fixed at compile time. Most statically typed languages enforce this by requiring you to declare all variables with their datatypes before using them. Java and C are statically typed languages.

dynamically typed language

A language in which types are discovered at execution time; the opposite of statically typed. VBScript and Python are dynamically typed, because they figure out what type a variable is when you first assign it a value.

strongly typed language

A language in which types are always enforced. Java and Python are strongly typed. If you have an integer, you can't treat it like a string without explicitly converting it.

weakly typed language

A language in which types may be ignored; the opposite of strongly typed. VBScript is weakly typed. In VBScript, you can concatenate the string '12' and the integer 3 to get the string '123', then treat that as the integer 123, all without any explicit conversion.

So Python is both *dynamically typed (because it doesn't use explicit data type declarations) and strongly typed (because* once a variable has a data type, it actually matters).

IDENTIFIER

- Name for variable, functions, class, modules and other objects
- Begin with alphabets(A - Z, a - z) and with numbers, letters or underscore.
- Special characters not allowed
- Case Sensitive
- Class Name starts with Capital Letter

- Multiline comments with “”” or ‘’’
- Comments begin with #(beginning or sideways)
- Here block is identified with indentation and not with {}
- Lengthy statements shall be separated with \
- Statements within {} [] () no need with \
- No typedeclaration

Assignments

- a = 10
- Pi = 3.14
- name = ‘murali’
- a = 10; b =20
- a,b = 10, 20
- a = b = c = 5

Data Types

- Numeric
- String
- List
- Tuple
- Dictionary
- Boolean

Numbers

- int
- float
- complex
- bool
- string
- bytes

+	Addition	Adds values on either side of the operator
-	Subtraction	Subtracts right hand operand from left hand operand
*	Multiplication	Multiplies values on either side of the operator
/	Division	Divides left hand operand by right hand operand
**	Exponent	Performs expo-nential (power) calculation on operators
//	Floor Division	The division of operands where the result is the quotient in which the digits after the decimal point are removed.
%	Modulo Division	Remainder of the division

```
>>> a = 20
>>> b = 10
>>> a + b
30
>>> a - b
10
>>> a * b
200
>>> a / b
2.0
>>> a ** 2
400
>>> a // b
2
>>> a % b
0
>>>
```

+=	Addition
-=	Subtraction
*=	Multiplication
/=	Division
**=	Exponent
//=	Floor Division
%=	Modulo Division

int()	(float/numeric string) (numeric string , base)
float()	(int/numeric strinc) (int)
complex()	(int/float)
bool()	(int/float)
bytes()	(int)
str()	(int/float/bool)
chr()	(int)

not Boolean NOT

== equal to

!= not equalto

< less than

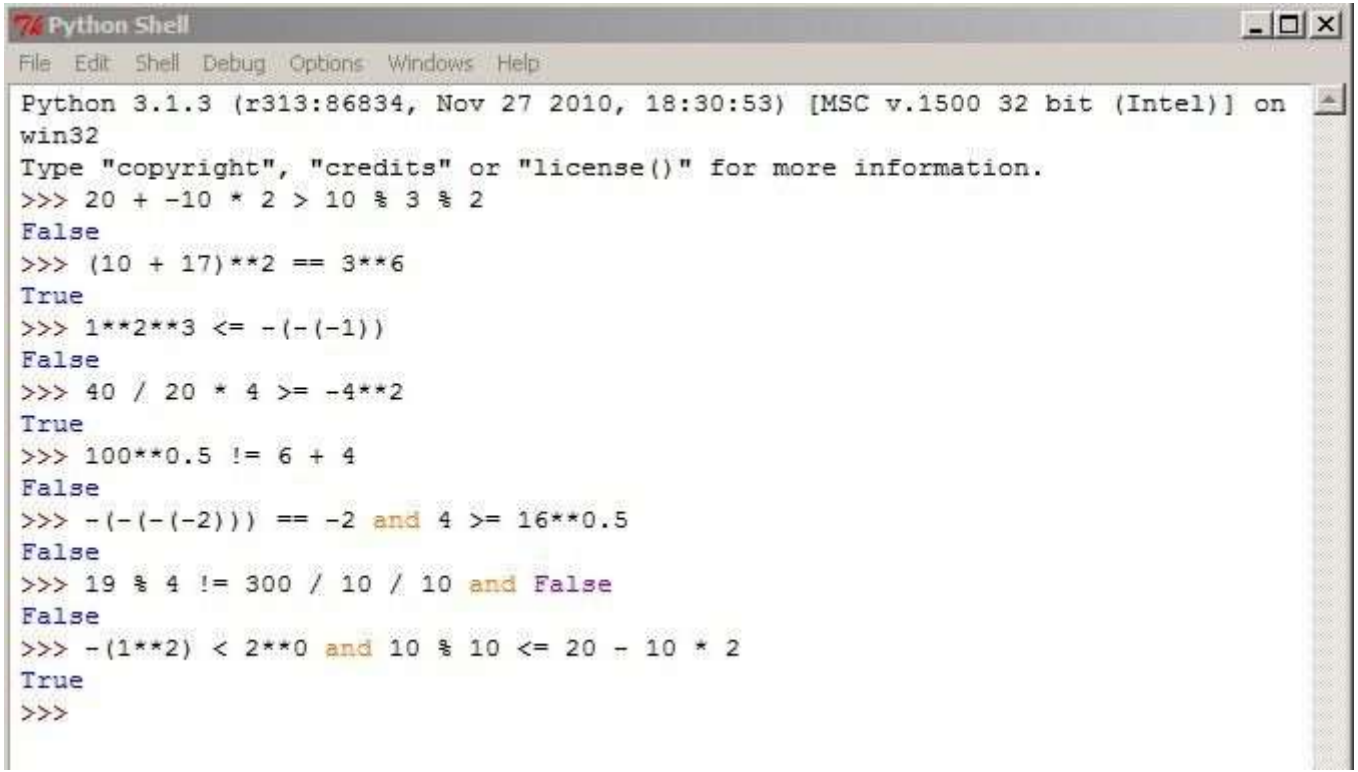
<= less than oreq

> greater than

>= greater than ore

<< >> & | ^ ~

[bit wise operators]


Python Shell
File Edit Shell Debug Options Windows Help
Python 3.1.3 (r313:86834, Nov 27 2010, 18:30:53) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 20 + -10 * 2 > 10 % 3 % 2
False
>>> (10 + 17)**2 == 3**6
True
>>> 1**2**3 <= -(-(-1))
False
>>> 40 / 20 * 4 >= -4**2
True
>>> 100**0.5 != 6 + 4
False
>>> -(-(-(-2))) == -2 and 4 >= 16**0.5
False
>>> 19 % 4 != 300 / 10 / 10 and False
False
>>> -(1**2) < 2**0 and 10 % 10 <= 20 - 10 * 2
True
>>>

Control Structures in Python

Control Structures

if *condition*:
    statements
[elif *condition*:
    statements] ...
else:
    statements

while *condition*:
    statements

for *var* in *sequence*:
    statements

break
continue

Important Note: Indentation

- In Python, all the statements indented by the same number of character spaces after a programming construct are considered to be part of a single block of code.

- Python uses indentation as its method of grouping statements.

If - Structures

- The **if** statement of Python is similar to that of other languages.

```
if expression:

statement(s)
```

Eg:

```
a=int(input("enter a="))
if (a%2==0):
    print "a is Even=",a
```

Output:

```
enter a=10
a is Even=10
```

If – else Structures

- The **if** statement of Python is similar to that of other languages.

```
if expression:
      statement(s)
else:
      statement(s)
```

Eg:

```
a=int(input("enter a:"))
if (a%2==0):
    print "a is even=",a
else:
     print "a is Odd=",a
```

Output:

enter a=5

a is Odd=5

If – elif-else Structures

```
if expression1:
    statement(s)
elif expression2:
    statement(s)
elif expression3:
    statement(s)
else:
    statement(s)
```

Eg:

```
a=int(input("enter a:"))
if (a==0):
    print "a is zero"
elif (a>0):
    print "a is Positive"
else:
    print " a is Negative"
```

while Structures

- The **while** loop continues until the expression becomes false.
- The expression has to be a logical expression and must return either a *true* or a *false* value

```
while expression:
    statement(s)
```

```
count = 0
while (count < 5):
    print 'The count is:', count
    count = count + 1

print "Good bye!"
```

Output:

```
The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
Good bye!
```

range()

creates a list of numbers in a specified range

range([start,] stop[, step]) -> list of integers

•When step is given, it specifies the increment (or decrement).

```
>>> range(5)
[0, 1, 2, 3, 4]

>>> range(5, 10)
[5, 6, 7, 8, 9]

>>> range(0, 10, 2)
[0, 2, 4, 6, 8]
```

The for loop

A **for loop** performs the same statements for each value in a list

for iterating_var in sequence:
statements(s)

- 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 statements(s) block is executed until the entire sequence is exhausted.

The for loop

Example:1

```
for n in range(1, 4):
    print "This is the number", n
```

OUTPUT:

```
This is the number 1
This is the number 2
This is the number 3
```

The for loop uses a variable (in this case, n) to hold the current value in the list

Example:2

```
for letter in 'Python’:
        print 'Current Letter :', letter
```

OUTPUT:

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : h
Current Letter : o
Current Letter : n
```

The *break* Statement:

•The **break** statement in Python terminates the current loop and resumes execution at the next statement

• just like the traditional break found in C.

•The most common use for break is when some external condition is triggered requiring a hasty exit from a loop.

• The **break** statement can be used in both *while* and *for* loops.

Eg:

```
for letter in 'Python‘:
    if letter == 'h':
        break
    print 'Current Letter :', letter
```

Output:

```
Current Letter : P
Current Letter : y
Current Letter : t
```

The *continue* Statement:

•The **continue** statement in Python returns the control to the beginning of the while loop.

•The **continue** statement rejects all the remaining statements in the current iteration of the loop and moves the control back to the top of the loop.

•The **continue** statement can be used in both *while* and *for* loops.

Eg:

```
for letter in 'Python‘:
    if letter == 'h':
        continue
    print 'Current Letter :', letter
```

Output:

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : o
Current Letter : n
```

The *pass* Statement:

- The **pass** statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.
- The **pass** statement is a *null* operation;
- nothing happens when it executes. The **pass** is also useful in places where your code will eventually go, but has not been written yet.

Eg:

```
for letter in 'Python':
      if letter == 'h':
         pass
         print 'This is pass block'
      print 'Current Letter :', letter

print "Good bye!"
```

Output:

```
Current Letter : P
Current Letter : y
Current Letter : t
This is pass block
Current Letter : h
Current Letter : o
Current Letter : n
Good bye!
```

Eg:2

```
for value in [3, 1, 4, 1, 5, 9, 2]:
    print "Checking", value
    if value > 8:
        print "Exiting for loop"
        break
    elif value < 3:
        print "Ignoring"
        continue
print "The square is", value**2
```

OUTPUT:

```
Checking 3
Checking 1
Ignoring
Checking 4
Checking 1
Ignoring
Checking 5
Checking 9
Exiting for loop
The square is 81
```

THANK YOU