Python Fundamentals

Python Character Set

Character set is the set of valid characters that a language can recognize. A character represents any letter, digit or any other symbol. Python has the following character sets:

- Letters – A to Z, a to z
- Digits – 0 to 9
- Special Symbols - + - * / etc.
- Whitespaces – Blank Space, tab, carriage return, newline, formfeed
- Other characters – Python can process all ASCII and Unicode characters as part of data or literals.

Tokens

In a passage of text, individual words and punctuation marks are called tokens lexical unite or lexical elements. The smallest individual unit in a program is called token.

Python has the following tokens:

(i) Keyword (ii) Identifiers
(iii) Literals (iv) Operators
(v) Punctuators

Keywords

Used to give special meaning to the interpreter and are used by Python interpreter to recognize the structure of program. These are reserved for special purpose and must not be used as normal identifier names. E.g. int, float etc.

Identifiers

Identifiers are the names given to different program elements like variable, objects, classes, function, list, dictionaries etc. It consists of letters and digits in any order except that the first character must be a letter, the underscore (_) counts as a character. Identifiers are unlimited in length. Python is a case sensitive programming language. Thus 'Value' and 'value' are two different identifiers in Python. All characters are significant.

Rules for Identifier:

- It consist of letters and digits in any order except that the first character must be a letter.
- The underscore (_) counts as a character.
- Spaces are not allowed.
- Special Characters are not allowed. Such as @,$ etc.
- An identifier must not be a keyword of Python.
- Variable names should be meaningful which easily depicts the logic.

Valid Identifiers	Invalid Identifiers
V, x1	4th (The first character must be a letter)
Sum-stud, TABLE	Order- no (Illegal character -)
Name, area, sum_1, _CH	Error flag (Illegal character Blank space)

Example: Out of the following, find those identifiers, which cannot be used for naming Variable or Functions in a Python program:

Total*Tax, While, class, switch, 3rdRow, finally, Column31, _Total

Following are cannot be used for naming Variable or Functions in a Python program:

Total*Tax, class, 3rdRow, finally

print ( ) statement:

To print or display output, Python provides print( ) function. We can print( ) as

print (<objects to be printed>….>

when we wrote

print (“Hello World!”)

it printed :

Hello World!

When we write:

print (‘Hello World!’)

it will print Hello World!

Note: We can enclose string either in double quotes or in single quotes but opening and closing quote should be of same type.

When print() statement is executed, the values are separated by space between them.

For example:

```
print ('values',10,20,30)
```

Output

values 10 20 30

We can also specify some other string as the separator using the sep argument of the print() function.

For example:

```
print ('values',10,20,30, sep="*")
print ('values',10,20,30, sep ="-*-")
print ('values',10,20,30, sep="\t")
print ('values',10,20,30, sep="\n")
```

Output:

```
values*10*20*30
values-*-10-*-20-*-30
values	10	20	30
values
10
20
30
```

Python multiple print statements without printing on the next line.

```
print ("value",10,20,30,end=" ")

print ("value",10,20,30)
```

Output

value 10 20 30 value 10 20 30

Literals

Literals (constants) are data items that have a fixed value. Python allows several kinds of literals:

(i) String literals (ii) Numeric literals (iii) Boolean literals
(iv) Special Literal None (v) Literals Collections

String literals

Text enclosed in quotes form a string literals in Python. E.g. ‘a’, ‘abc’, “abc”. Following are some valid string literals:

‘John’, “Tom”, ‘Class 12’, “200453A”, ‘12345’

Note: String literals are a sequence of characters surrounded by quotes: single, double or triple.

Python allows us to have 2 string types:

- **Single-line string**

String created by enclosing text in single quotes, a double quotes are normally single-line string, i.e., they must be terminated in one line. E.g. A variable a='John'.

- **Multi-line string**

In multi-line string, text is spread across multiple lines as one single string. It can be created in 2 ways:

(a) By adding a backslash at the end of normal single quote or double quote string. In normal strings, just add a backslash before pressing 'enter' to continue.

```
e.g.  a="class\
      XI"
```

(b) By typing the text in triple quotation marks.

```
e.g. a="""class
     XI"""
     B='''class
     XII'''
```

Escape Sequence

Python allows to have certain non-graphic characters in string values. Such characters cannot be typed directly from keyboard.

e.g. backspace, tabs, carriage return etc.

These can be represented by escape sequence. An escape sequence is represented by backslash followed by one or more characters. Following are the list of escape sequence:

```
\"    - Double quotes
\'    - Single quotes
\\    - Backslash
\a    - Bell
\b    - Backspace
\n    - New line
\r    - Carriage return
\t    - Horizontal tab
\v    - Vertical tab
```

Size of String

Python determines the size of a string as the count of characters in the string.

e.g. sizeof "abc"=3

But, if our string literals have an escape sequence contained within it, then make sure to count the escape sequence as one character.

e.g. '\\' size is 1 (escape sequence)
"\ab" size is 2
"abdul\' s
"Seema\'s pen" size is 11

For multi-line string with triple quote, while calculating the size, the EOL(End Of Line), the character at the end is also counted.

```
a="""X
y
z"""                    has size 5.
```

For multi-line string with single quote and backslash character at the end of line, while calculating the size, backslash is not counted.

```
a="c\
d\
e"
```

Numeric literals

Numeric literals in Python can belong to any of the following 4 types:

- **int (signed integer)**

  Often called integers. These are positive or negative numbers with no decimal point.

- **long (long integer)**

  long integers or longs are integers of unlimited size, written like integers and followed by an upper or lowercase 'l'.

- **float (floating point real values)**

  float represents real numbers and are written with decimal point, dividing the integer and fractional parts.

- **complex (complex number)**

  Are in the form of a + bj;

  e.g.

```
x=2 + 0j
print x.real
print x.imag
```

2.0
0.0

e.g

```
x=7 + 8j
print x.real, " ",x.imag

7.0   8.0
```

Python allows 3 types of integer literals:

(a) **Decimal Integer Literal**: An integer literal consisting of sequence of digits taken to be a decimal integer literal unless it begins with 0 (zero).

(b) **Octal Integer Literal**: A sequence of digits starting with 0 is taken as octal integer literal.
e.g. Decimal 8 is 010.

(c) **Hexadecimal Integer Literal**: A sequence of digits preceded by OX or ox is taken to be hexadecimal integer literal.

(d) **Floating Point Literal**: A number with a fractional part or decimal point consists of sign (+,-), sequence of decimal digits with a dot such as
0.0609, 15.0609

These numbers can also be used to represent a number in engineering and scientific notation

$6.09 * 10^{-2}$, $1.50609 * 10^{1}$

Note: Exponent form – A real literal in exponent form consists of 2 parts: mantissa and exponent. E.g. 6.09 E0(-2) is exponent form of 0.0609.

Boolean literals

Used to represent one of the two boolean values, that is True and False.

For example:

```
>>>bool_1=(6>10)
>>>print (bool_1)
False
```

Special literals

- **None**

It is used to indicate something that has not been created yet or to signify the absence of values in a situation. Python doesn't display anything when we give a command to display the value of a variable containing value as None.

For example:

```
>>>value1=20
>>>value2=None
>>>value1
```