Python Variables

- ☐ Python is a '**dynamically typed**' language. We do not need to declare variable types before using them.
- ☐ A variable is created when we assign a value to it first time.
- ☐ Rules to create a variable in python:
  - A variable name can contain only numbers, letters and underscore(_).
  - A variable name must starts with a letter or underscore.
  - A variable name cannot start with a number.
  - Variable names are case-senstive.

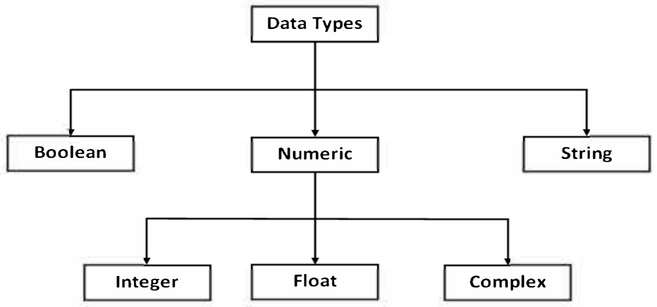
Python Variables

- ☐ **my_variable**, **_myVariable**, **_message2**, **message_5**, **a_b_c**, **abc** are all valid variable names.
- ☐ **myVariable@**, **2_abc**, **#message**, **message*** are not valid variable names.

```
>>>myVariable = 4
>>>a_ = 0.25
>>>_message2 = "Hello World"
>>>myVariable@ = 123
  File "<stdin>", line 1
   myVariable@ = 123
              ^
SyntaxError: invalid syntax
```

Data Type

- ☐ A data type is an attribute of data which tells the compiler or interpreter how the programmer wants to use the data.
- ☐ In python every thing is an object and data type is associated with object.
- ☐ Every value in python has a datatype. Data types are classes and objects belong to their respected these classes.



Basic Data Types in Python

Numeric Data Type

Real numbers (Integers and floats) and complex numbers are numeric data type and are defined as int, float and complex class in python.

```
>>>4
4
>>>type(4)
<class 'int'>
>>>a = 4.0
>>>type(a)
<class 'float'>
>>>2E-3
0.002
>>>type(2E-3)
<class 'float'>
```

```
>>>3+5j
3+5j
>>>type(3+5j)
<class 'complex'>
>>>myVariable = j
Traceback (most recent call last):
NameError: name 'j' is not defined
>>>myVariable = 1j
>>>type(myVariable)
<class 'complex'>
>>>
```

Boolean Data Type

- ☐ Boolean values are two constant objects True and False.
- ☐ Boolean data type is used to represent the truth value.
- ☐ In numerical context boolean variable behave as integer 0 and 1.

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

String Data Type

- ☐ A string is a sequence of characters.
- ☐ A string object can be created using either single quotes (' ') or double quotes(" ").A multi-line string is created using triple quotes('''or """).

```
>>>a = 'Hello World'
>>>print(a)
Hello World
>>>type(a)
<class 'str'>
>>>b = "Hello  World"
>>>print(b)
Hello World
>>>type(b)
<class 'str'>
```

```
>>>c = '''Hello
... World'''
>>>print(c)
Hello
World
>>>type(c)
<class 'str'>
>>>
```

String Methods

Method	Description
upper()	Returns the uppercase version of string
lower()	Returns the lowercase version of string
capitalize()	Returns a new string where the first letter is capitalized and rest are lowercase
title()	Return a string where the first letter of each word is capital and all other are lowercase
replace(old,new,max)	Returns a sting where occurrences of the sub-string old are replaced with the sub-string new, max limits the number of replacements and is optional

String Methods

swapcase()	Return a new string with case of each letter swapped

```
>>>str1 = 'hello World'
>>>print(str1)
hello World
>>>print(str1.upper())
HELLO WORLD
>>>print(str1.lower())
hello world
>>>print(str1.capitalize())
Hello world
```

```
>>>print(str1.title())
Hello World
>>>print(str1.replace('o','OO'))
hellOO WOOrld
>>>print(str1.replace('o','OO',1))
hellOO World
>>>print(str1.swapcase())
HELLO wORLD
```

Input

The input() function in python reads a line from user, converts into string and retrun it.

Syntax:

input(prompt)

prompt: message before input (optional)

```
>>>user_input = input()
123
>>>print(user_input)
123
>>>type(user_input)
<class 'str'>
>>>
```

Type Casting

```
>>>user_input = input('Enter a numerical value: ')
Enter a numerical value: 123
>>>user_input = int(user_input)
>>>print(user_input)
123
>>>type(user_input)
<class 'int'>
>>>user_input = float(user_input)
>>>print(user_input)
123.0
>>>type(user_input)
<class 'float'>
>>>user_input = complex(user_input)
>>>print(user_input)
(123+0j)
```

Type Casting

- Type casting is the process of conversion of data type of the object using predefined function.
- Loss of data may occur in type casting, because, we enforce the object to specific data type.

Function	Description
int()	converts any data type to integer
float()	converts any data type to float
comlex(real,imag)	converts real number to complex number
bool()	converts any data type to boolean value
str()	converts numeric data type to string

Type Casting

```
>>>type(user_input)
<class 'complex'>
>>>user_input = bool(user_input)
>>>print(user_input)
True
>>>type(user_input)
<class 'bool'>
>>>user_input = str(user_input)
>>>print(user_input)
True
>>>type(user_input)
<class 'str'>
```

Output

In python we use print() function to display the output data on screen.

```
>>>a = 10
>>>b = 20
>>>print("The value of a is",a)
The value of a is 10
>>>print("The value of a is",a,"and value of b is",b)
The value of a is 10 and value of b is 20
>>>print("The value of a is {}
                and value of b is{}".format(a,b))
The value of a is 10 and value of b is20
>>>print(30,40,50,sep='@')
30@40@50
>>>
```

Formatted Output

```
>>>print("value1: %3d and value2: %8.3f" %(5,3.14))
value1:   5 and value2:    3.140
>>>print("value1: %3d and value2: %1.4f" %(5,3.14))
value1:   5 and value2: 3.1400
>>>print("value1: {0:3d} and
                value2: {1:8.3f}".format(5,3.14))
value1:   5 and value2:    3.140
>>>print("value1: {1:8.3f} and
                value2: {0:3d}".format(5,3.14))
value1:    3.140 and value2:   5
>>>print("value2: %3d" %(3.14))
value1:   3
>>>
```

Formatted Output

We use string modulo operator(%) to get formatted output.

Syntax:

%[Width].[Precision]Type

Width: total number of digits in formated output value including decimal point in case of floating point value

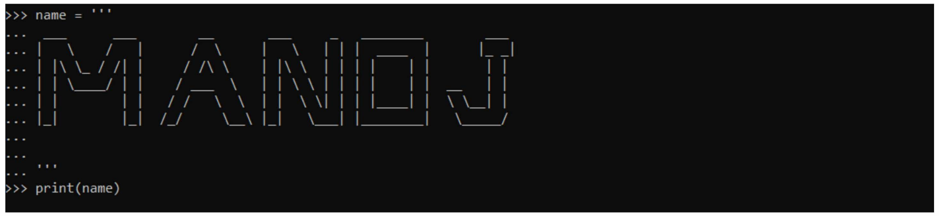
Precision: number of digits after decimal point

Type: data type (**d** for integer values and **f** for floating point values)

Note: If Width < Precision, then it will print integer part followed by decimal part upto specified precision.

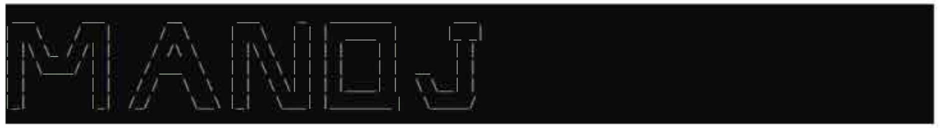
Multi-line String Output

Multi-line string output using print() function.



```
>>> name = '''
...
...
...
...
...
...
...
...
...   '''
>>> print(name)
```

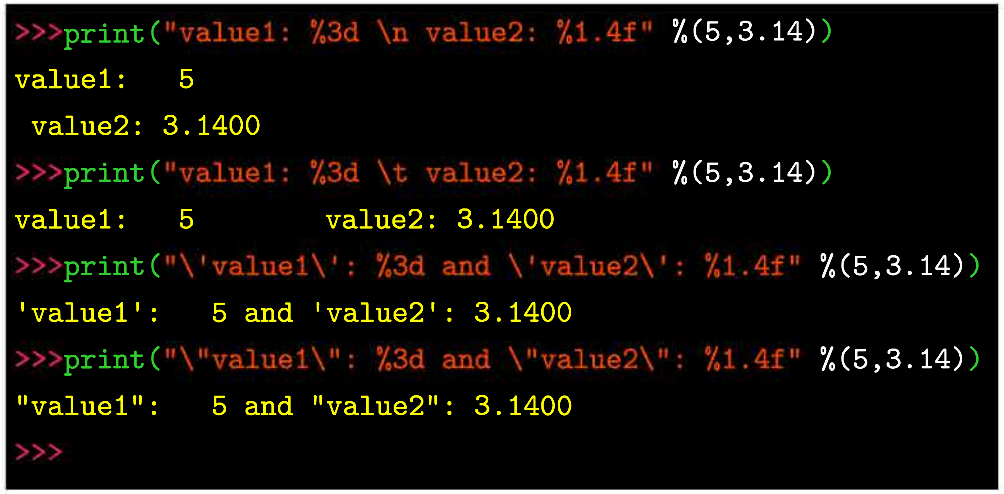
Output



Escape Sequences with String

Sequence	Description
\\	Prints one backslash
\'	Prints a single quote
\"	Prints a double quote
\n	Moves cursor to beginning of next line
\t	Moves cursor forward one tab step

Escape Sequences with String



```
>>>print("value1: %3d \n value2: %1.4f" %(5,3.14))
value1:   5
 value2: 3.1400
>>>print("value1: %3d \t value2: %1.4f" %(5,3.14))
value1:   5          value2: 3.1400
>>>print("\'value1\': %3d and \'value2\': %1.4f" %(5,3.14))
'value1':   5 and 'value2': 3.1400
>>>print("\"value1\": %3d and \"value2\": %1.4f" %(5,3.14))
"value1":   5 and "value2": 3.1400
>>>
```