Unit –1
Python Fundamentals

Python Basics – Variables

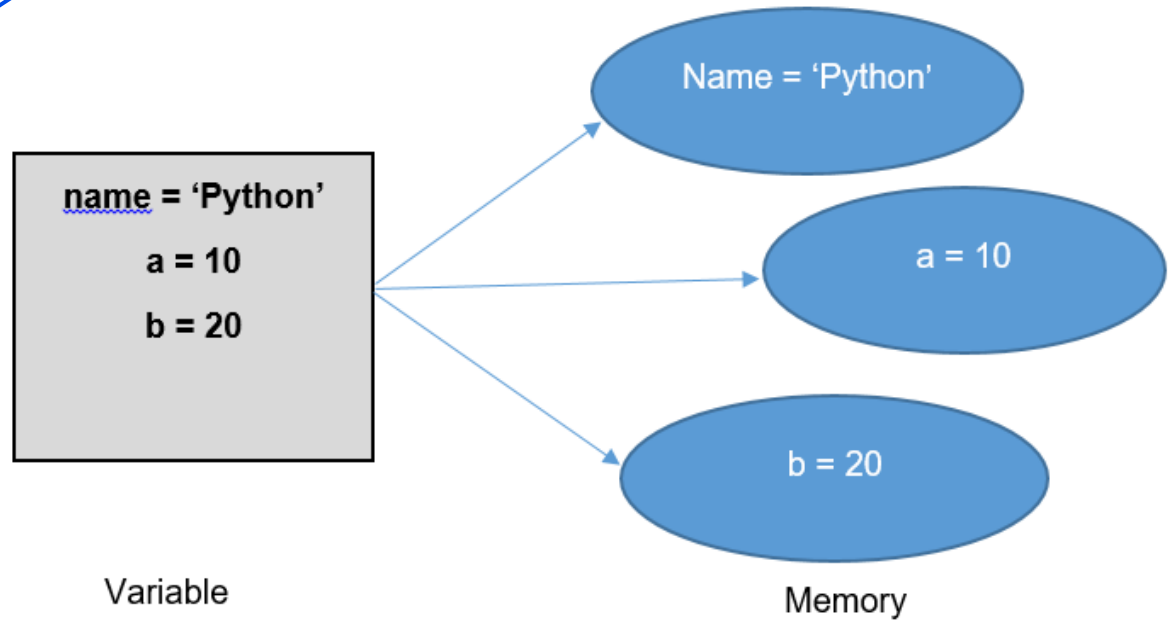
A variable is a container for a value. It can be assigned a name; you can use it to refer to it later in the program.

Rules for Variable Names

- Variable name cannot start with a number
- It cannot contain spaces, use _ instead
- Names can not contain any of these symbols (:'",<>/?|\!@#%^&*~-+)
- Avoid using Python built-in keyword Variables names are case-sensitive

Assigning Values to Variable

- Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables.



Multiple Assignment

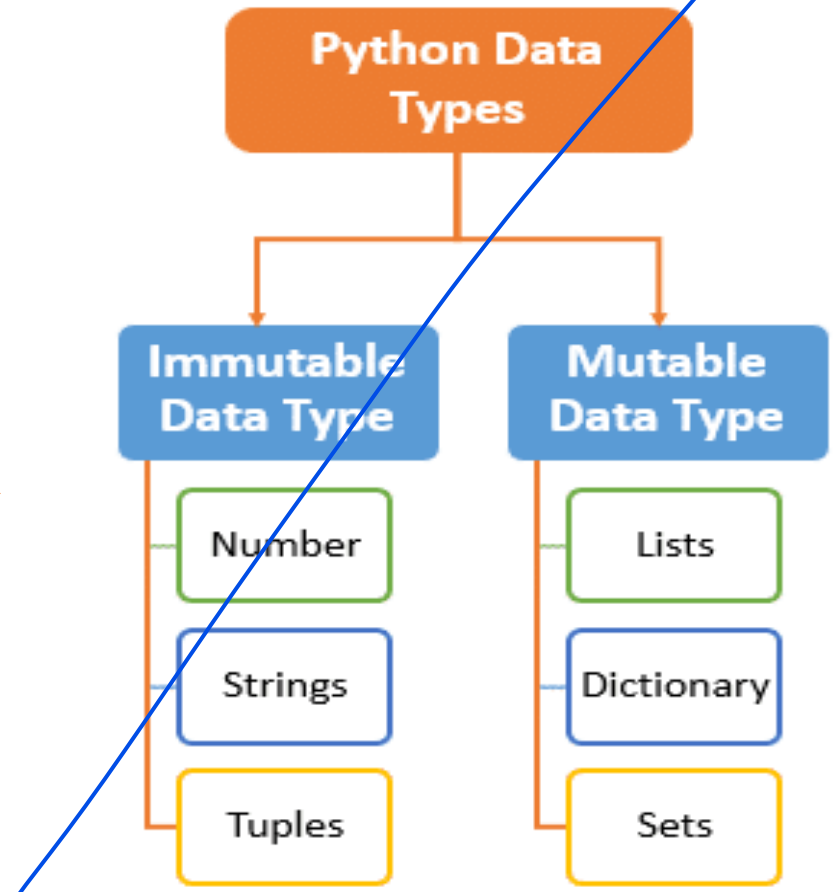
- Python allows you to assign a single value to several variables simultaneously. For example –

- Here, an integer object is created with the value 1, and all three variables are assigned to the same memory location.

```
In [1]: count = 100                    #Integer Assignment
        kilometers = 1000.0            # Float Assignment
        name = 'Rahul'                 # String Assignment

In [2]: print(count,kilometers,name)

        100 1000.0 Rahul

In [4]: a,b,c=1,2,"john"
        print(a,b,c,sep=',')

        1,2,john

In [5]: a = b = c = 1

In [6]: print(a,b,c)

        1 1 1
```

Standard Data Types

- Numerical
  - Int
  - Float
  - Complex no.
- String
- Tuples
- Lists
- Dictionary
- Sets



pyhon3.png (681×374) (intellipaat.com)

Python Numbers

- Number data types store numeric values. Number objects are created when you assign a value to them.

```
In [7]: a = 10

In [8]: a
Out[8]: 10

In [12]: # Multiple Assignment
         b,c,d = 11,12,13

In [13]: print(a,b,c,d)
         10 11 12 13
```

- You can also delete the reference to a number object by using the del statement. The syntax of the del statement is –

```
In [14]: #You can also delete the reference to a number object by using the del statement.
         del d
```

Python Numbers

- You can also delete multiple objects by using the del statement. For example

```
In [15]: d
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-15-e983f374794d> in <module>
----> 1 d

NameError: name 'd' is not defined
```

Python supports three different numerical types –

- int (signed integers)
- float (floating point real values)
- complex (complex numbers)

```
In [16]: del b,c

In [17]: b
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-17-89e6c98d9288> in <module>
----> 1 b

NameError: name 'b' is not defined
```

Numerical Data Types

int	float	complex
10	0	3.14j
100	15.2	45.j
-786	-21.9	9.322e-36j
80	32.3+e18	.876j
-490	-90	-.6545+0J
-0x260	-3.25E+101	3e+26J
0x69	70.2-E12	4.53e-7j

- A complex number consists of an ordered pair of real floating-point numbers denoted by x + yj, where x and y are the real numbers and j is the imaginary unit.

```
In [18]: a=32.3e18
         type(a)

Out[18]: float

In [19]: b=3.14j
         type(b)

Out[19]: complex

In [20]: print(a,b,sep='\n')

         3.23e+19
         3.14j
```

Python Strings

- Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows either pair of single or double quotes.

- The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator.

- **For example –**

```
In [21]: str1 = 'Python is a programming language'
```

Python Strings

```
In [22]: print(str1)

         Python is a programming language

In [23]: print(str1[0])

         P
```

```
In [24]: print(str1[2:10])

         thon is

In [25]: print(str1[6:18])

          is a progra
```

```
In [27]: print(str1[:])

         Python is a programming language
```

Deleting/Updating from a String

- In Python, Updating or deletion of characters from a String is not allowed. This will cause an error because item assignment or item deletion from a String is not supported.

```
In [32]: #Updation of a character:
         str1[0]='A'

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-32-6f42197499b8> in <module>
      1 #Updation of a character:
----> 2 str1[0]='A'

TypeError: 'str' object does not support item assignment
```

Escape Sequences

```
In [33]: #Deletion of a character:
         del str1[1]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-33-b0bad0c6f509> in <module>
      1 #Deletion of a character:
----> 2 del str1[1]

TypeError: 'str' object doesn't support item deletion
```

```
In [34]: #Updating Entire String:
         str1="New String"
         print(str1)
```

```
New String
```

```
In [36]: # Deletion of Entire String
         del str1
```

Escape Sequence	Description	Example	Output
\\	Prints Backslash	print "\\"	\
\`	Prints single-quote	print "\'"	'
\"	Pirnts double quote	print "\""	"
\a	ASCII bell makes ringing the bell alert sounds (eg. xterm)	print "\a"	N/A
\b	ASCII backspace (BS) removes previous character	print "ab" + "\b" + "c"	ac
\f	ASCII formfeed (FF)	print "hello\fworld"	hello world
\n	ASCII linefeed (LF)	print "hello\nworld"	hello world

Escape Sequences

\N{name}	Prints a character from the Unicode database	print u"\N{DAGGER}"	†
\r	ASCII carriage return (CR). Moves all characters after (CR) the the beginning of the line while overriding same number of characters moved.	print "123456\rXX_XX"	XX_XX6
\t	ASCII horizontal tab (TAB). Prints TAB	print "\t* hello"	* hello
\t	ASCII vertical tab (VT).	N/A	N/A
\uxxxx	Prints 16-bit hex value Unicode character	print u"\u041b"	Л
\Uxxxxxxxx	Prints 32-bit hex value Unicode character	print u"\U000001a9"	Σ
\ooo	Prints character based on its octal value	print "\043"	#
\xhh	Prints character based on its hex value	print "\x23"	#
	LinuxConfig.org		

String Formatting

- Strings in Python can be formatted with the use of format() method which is very versatile and powerful tool for formatting of Strings.

- Format method in String contains curly braces {} as placeholders which can hold arguments according to position or keyword to specify the order.

```
In [40]: # Default order
         String1 = "{} {} {}".format('Python', 'Programming', 'Language')
         print("Print String in default order: ")
         print(String1)
```

```
Print String in default order:
Python Programming Language
```

String Formatting

In [41]:
```
# Positional Formatting
String1 = "{1} {0} {2}".format('Python', 'Programming', 'Language')
print("\nPrint String in Positional order: ")
print(String1)
```

```
Print String in Positional order:
Programming Python Language
```

In [42]:
```
# Keyword Formatting
String1 = "{a} {e} {c}".format(a='Python', e='Programming', c='Language')
print("\nPrint String in order of Keywords: ")
print(String1)
```

```
Print String in order of Keywords:
Python Programming Language
```

Python List

- Lists are the most versatile of Python's compound data types.
- A list contains items separated by commas and enclosed within square brackets ([]). To some extent, lists are like arrays in C.

```
In [43]: L=[1,2,3,"abes",12.5]

In [44]: L
Out[44]: [1, 2, 3, 'abes', 12.5]

In [45]: L[0:3]
Out[45]: [1, 2, 3]
```

Python List

- One difference between them is that all the items belonging to a list can be of different data type.
- The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1. The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator. **For example –**

```
In [52]: list1 = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
         small = [123, 'john']

In [54]: print(small * 2)        # Prints list two times
         print(list1 + small)    # Prints concatenated lists

         [123, 'john', 123, 'john']
         ['abcd', 786, 2.23, 'john', 70.2, 123, 'john']
```

Python List

```
In [46]: L[0]=10

In [47]: L
Out[47]: [10, 2, 3, 'abes', 12.5]
```

```
In [50]: l
Out[50]: ['12', '34', '45', '56', '67']

In [51]: num=int(input("Enter a number"))
         print(num,"",sep='.')
Enter a number12
12.
```

```
In [48]: #Write a program to take input a sentence and split it into word
         str=input("Enter any sequence")
Enter any sequence12 34 45 56 67

In [49]: l=str.split()
```

Tuples

- A tuple is another sequence data type that is like the list.
- A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.

```
In [55]: tuple1 = ( 'abcd', 786 , 2.23, 'john', 70.2  )
         tuple2 = (123, 'john')

In [58]: print(tuple1)                    # Prints the complete tuple

         ('abcd', 786, 2.23, 'john', 70.2)
```

Tuples

```
In [59]: print(tuple1[0])          # Prints first element of the tuple
abcd

In [60]: print(tuple1[1:3])        # Prints elements of the tuple starting from 2nd till 3rd
(786, 2.23)

In [61]: print(tuple1[2:])         # Prints elements of the tuple starting from 3rd element
(2.23, 'john', 70.2)

In [62]: print(tuple2 * 2)         # Prints the contents of the tuple twice
(123, 'john', 123, 'john')

In [64]: print(tuple1 + tuple2)    # Prints concatenated tuples
('abcd', 786, 2.23, 'john', 70.2, 123, 'john')
```

Tuples

- The following code is invalid with tuple, because we attempted to update a tuple, which is not allowed. Similar case is possible with lists –

```
In [66]: tuple1 = ( 'pqrs', 123 , 2.14, 'python', 98.6  )
         list1 = [ 'pqrs', 123 , 2.14, 'python', 98.6  ]

In [67]: tuple1[2] = 1000    # Invalid syntax with tuple
         list1[2] = 1000     # Valid syntax with list

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-67-ea927ce875cf> in <module>
----> 1 tuple1[2] = 1000    # Invalid syntax with tuple
      2 list1[2] = 1000     # Valid syntax with list

TypeError: 'tuple' object does not support item assignment
```

Python Dictionary

- Python's dictionaries are kind of hash table type.
- They work like associative arrays or hashes found in Perl and consist of key-value pairs.
- Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).

```
In [71]: dict1 = {}
         dict1['one'] = "This is Python"
         dict1[2]     = "This is Java"

         dict2 = {'name': 'rohit','id':1234, 'dept': 'technical'}
```

```
In [72]: print(dict1['one'])       # Prints value for 'one' key
         This is Python

In [73]: print(dict1[2])          # Prints value for 2 key
         This is Java
```

Python Dictionary

```
In [74]: print(dict2)          # Prints complete dictionary
         {'name': 'rohit', 'id': 1234, 'dept': 'technical'}

In [75]: print(dict2.keys())   # Prints all the keys
         dict_keys(['name', 'id', 'dept'])

In [76]: print(dict2.values()) # Prints all the values
         dict_values(['rohit', 1234, 'technical'])
```

Data Type Conversion

Sr.No.	Function & Description
1	**int(x [,base])** Converts x to an integer. base specifies the base if x is a string.
2	**long(x [,base])** Converts x to a long integer. base specifies the base if x is a string.
3	**float(x)** Converts x to a floating-point number.
4	**complex(real [,imag])** Creates a complex number.
5	**str(x)** Converts object x to a string representation.

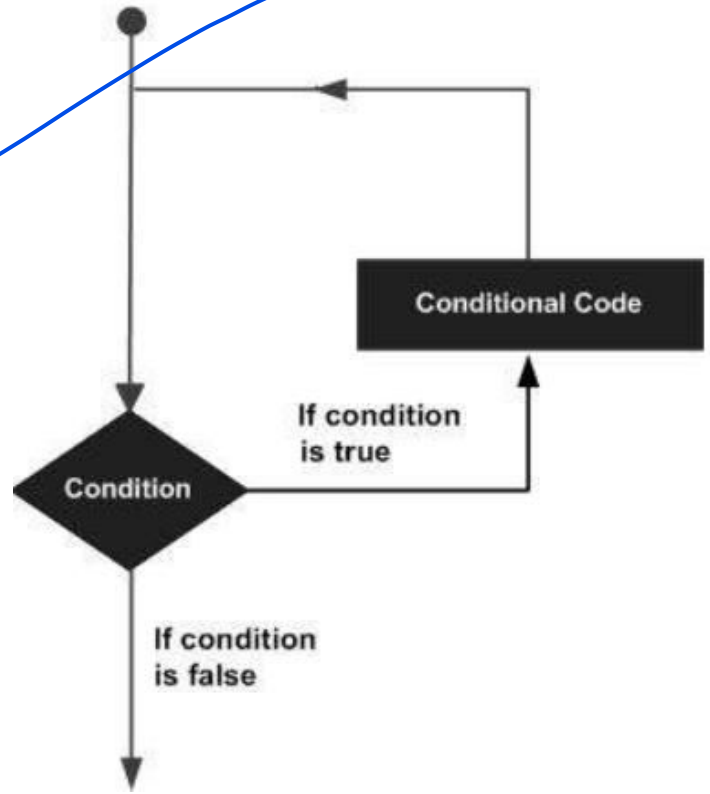
Data Type Conversion

6	**repr(x)** Converts object x to an expression string.
7	**eval(str)** Evaluates a string and returns an object.
8	**tuple(s)** Converts s to a tuple.
9	**list(s)** Converts s to a list.
10	**set(s)** Converts s to a set.
11	**dict(d)** Creates a dictionary. d must be a sequence of (key,value) tuples.
12	**frozenset(s)** Converts s to a frozen set.
13	**chr(x)**
14	**unichr(x)** Converts an integer to a Unicode character.
15	**ord(x)** Converts a single character to its integer value.
16	**hex(x)** Converts an integer to a hexadecimal string.
17	**oct(x)** Converts an integer to an octal string.

Python Loops

In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on. There may be a situation when you need to execute a block of code several number of times.

The diagram illustrates a loop statement −



https://www.tutorialspoint.com/python/python_loops.htm

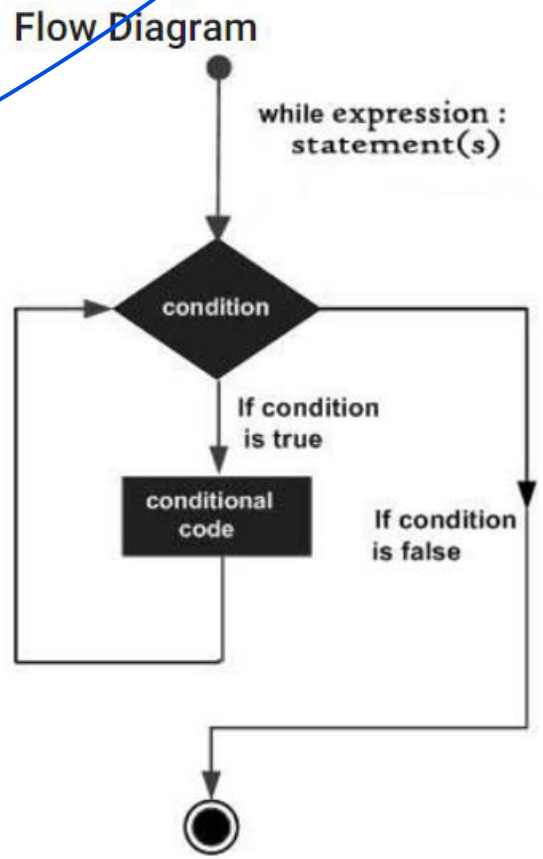
Types of Loops

1. While Loop

A **while** loop statement in Python programming language repeatedly executes a target statement as long as a given condition is true.

Syntax :

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



https://www.tutorialspoint.com/python/python_loops.htm

Types of Loops

Example

```
count = 0
while (count < 9):
    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
The count is: 5
The count is: 6
The count is: 7
The count is: 8
Good bye!
```

Types of Loops

2. While Loop with else

Python supports to have an else statement associated with a loop statement. If the else statement is used with a while loop, the else statement is executed when the condition becomes false.

https://www.tutorialspoint.com/python/python_loops.htm

Types of Loops

Example

```
count = 0
while count < 5:
    print(count, " is  less than 5")
    count = count + 1
else:
    print(count, " is not less than 5")
```

Output

```
0  is  less than 5
1  is  less than 5
2  is  less than 5
3  is  less than 5
4  is  less than 5
5  is not less than 5
```

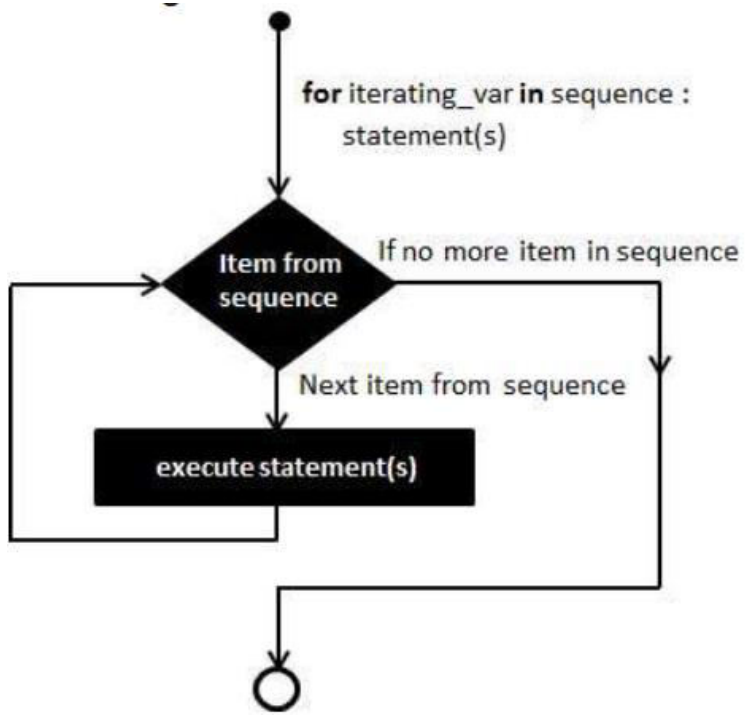
Types of Loops

1. For Loop

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

Syntax :

for iterating_var in sequence:
statements(s)



https://www.tutorialspoint.com/python/python_loops.htm

Types of Loops

Example

```
for letter in 'Python':     # First Example
    print('Current Letter :', letter)

fruits = ['banana', 'apple',  'mango']
for fruit in fruits:        # Second Example
    print('Current fruit :', fruit)

print("Good bye!")
```

Output

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : h
Current Letter : o
Current Letter : n
Current fruit : banana
Current fruit : apple
Current fruit : mango
Good bye!
```

Types of Loops

2. For Loop with else

Python supports to have an else statement associated with a loop statement. If the else statement is used with a for loop, the else statement is executed when the loop has exhausted iterating the list.

https://www.tutorialspoint.com/python/python_loops.htm

Types of Loops

Example

```
for num in range(10,20):      #to iterate between 10 to 20
    for i in range(2,num):     #to iterate on the factors of the number
        if num%i == 0:          #to determine the first factor
            j=num/i              #to calculate the second factor
            print('%d equals %d * %d' % (num,i,j))
        break    #to move to the next number, the #first FOR
    else:                   # else part of the loop
        print(num, 'is a prime number')
        break
```

Output

```
10 equals 2 * 5
12 equals 2 * 6
14 equals 2 * 7
16 equals 2 * 8
18 equals 2 * 9
```

References

- https://www.tutorialspoint.com/python/python_variable_types.htm
- https://www.edureka.co/blog/variables-and-data-types-in-python/
- https://data-flair.training/blogs/python-variables-and-data-types/
- https://www.programiz.com/python-programming/variables-datatypes