**Python Basics**

**Python**

* Python is an object oriented, interpreted, high-level programming language
* Python was created by Guido Van Rossom (Google Employee)
* The initial code was published as version 0.9 in 1991 and grew up to version 1.0 in January 1994
* Python is a case sensitive language

**Why the language is named Python?**

It is named after a BBC show called Monty’s Python Flying Circus [British Comedy Group – TV Show]

**Why Python?**

* Python is simple and easy to learn
* It has libraries for almost everything
* DJango : for Web Development
* Matplotlib : for Data Visualization
* It is reliable, efficient and time-saving
* It has a large, active, and helpful open-source community
* It offers career opportunities in various fields

**Where can we use Python?**

* Web Application
* DJango, Flask, Tornado
* Desktop Application
* PyGTK, Cocoa
* Machine Learning, Data Science and Deep Learning
* Sklearn, TensorFlow, Theano, Spark
* Hardware Programming
* Raspberry Pi

**Variables**

A Variable is a memory location where we can store values.

In Python, the data type will be identified according to the data we provide.

**Rules:**

1. A variable should start with a letter or an underscore
2. Cann’t start with numbers, special characters

There are two ways of assigning values to a variable:

1. Assigning a Single Value

i = 10, b = ‘Upendra’

1. Multiple Assignment

a = b = c = 10

x , y, z = 20, 30, 40

name, age = 25 × **wrong assignment**

**Python Tokens**

In Python, every logical line of code is broken down into components known as Tokens.

1. **Keywords:**

* Python Keywords are special reserved words
* They convey a special meaning to the compiler / interpreter
* Each keyword has a special meaning and a specific operations
* Never use it as variable

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| True | False | None | and | as | Asset |
| def | class | continue | break | else | finally |
| elif | del | except | global | for | if |
| from | import | raise | try | or | returns |
| pass | nonlocal | in | not | is | lambda |

1. **Identifiers**

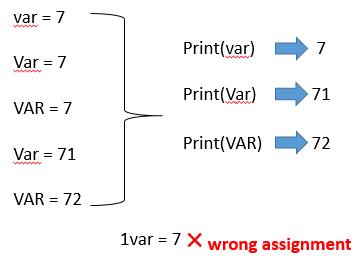
An identifiers is the name used to identify a variable, function, class, or object

**Rules :**

1. No special character, except underscore ( \_ ) can be used as an identifiers
2. Keywords should not be used as an identifiers
3. Python is case sensitive

e.g. “Var” and “var” are two different idenifiers

1. The first character of an identifier can be a alphabet or underscore ( \_ ) but not a digit



1. **Literals**

A literal is the raw data given to a variable.

**String Literals:**

Formed by enclosing a text within quotes, both single ( ‘ ) and double quotes ( “ ) can be used.

|  |  |
| --- | --- |
| **Input** | **Output** |
| name1 = “John” |  |
| name2 = “James” |  |
| print (name1) | John |
| print(name2) | James |
| text1 = ‘Hello \  World’  Print(text1) | Hello World |
| # used for long statements  multiline = ‘’’ str1  str2  str3 ‘’’  print(multiline) | str1  str2  str3 |

**Numeric Literals:**

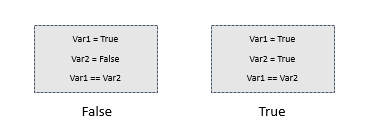
Formed by a character string of digits from 0-9, decimal points, and a plus (+) or minus (-) sign.

|  |  |  |  |
| --- | --- | --- | --- |
| **Int** | **Long** | **Float** | **Complex** |
| +ve and –ve numbers (integer) with no fractional parts  e.g. 100, 234 | An unlimited string of Integers followed by upper or lower case L  e.g. 23342424L | Real numbers with both Integer and fractional parts  e.g. -213.4 | Strings in the form of a+bj, where ‘a’ is the real part & ‘b’ is the imaginary part  e.g 3.14j |

1. In Python, the value of an Integer is not restricted by the number of bits, and it can expand to the limit of the available memory
2. No special arrangement is required for storing large numbers

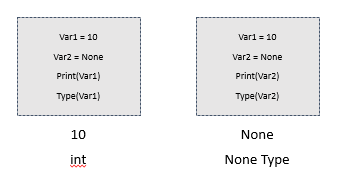
**Boolean Literals:**

It can be either TRUE or FALSE.



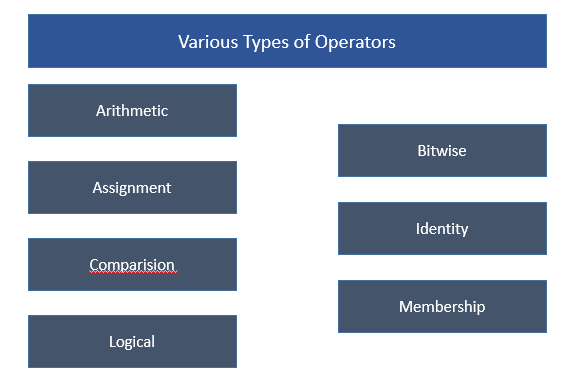
**Special Literals:**

There is a special literals in Python called **None** which means that the variable is yet to be initialized.



1. **Operators**

Operators are special symbols that are used to carry out arithmetic and logical operations.



**Arithmetic Operators:**

|  |  |
| --- | --- |
| + | Addition |
| - | Substation |
| \* | Multiplications |
| / | Division |
| % | Modulus (Gives Remainder) |
| \*\* | Exponentiation |

**Assignment Operators:**

Used to assign values to variable.

|  |  |
| --- | --- |
| **Operator** | **Operation** |
| = | x = 10 |
| += | x = x + 2 |
| - = | x = x - 29 |
| \*= | x = x \* 29 |
| /= | x = x / 13 |
| |= | x = x | 6 |

**Comparison Operators:**

Compare values and return either True or False.

|  |  |
| --- | --- |
| **Operator** | **Operation** |
| == | Equal |
| != | Not Equal |
| < | Less than |
| > | Greater than |
| >= | Greater than or Equal to |
| <= | Less than or Equal to |

**Logical Operators:**

Used to combine conditional statements.

|  |  |
| --- | --- |
| **Operator** | **Operation** |
| and | True if both statements are True |
| or | True if one of the statements is True |
| not | If True, then return False |

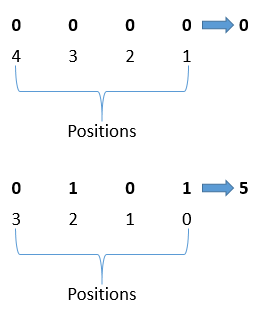
**Bitwise Operators:**

Used to compare binary numbers.

|  |  |
| --- | --- |
| **Operator** | **Operation** |
| & | AND |
| | | OR |
| ^ | X OR |
| ~ | NOT |
| << | LEFT SHIFT |
| >> | RIGHT SHIFT |

Binary Numbers : 0, 1

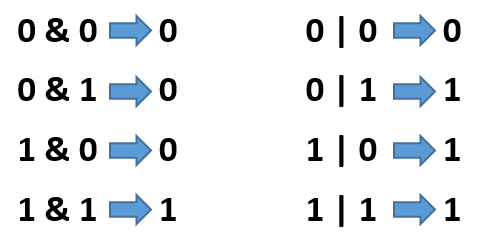
4 bit -> 4 binary digits

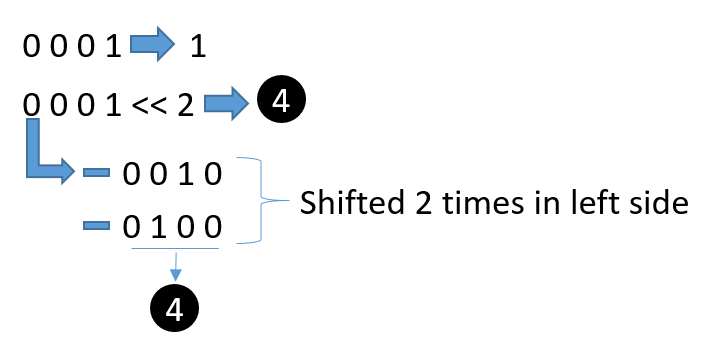


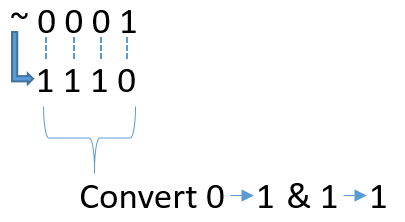
[ ( 2 ^ 3 ) \* 0 ] + [ ( 2 ^ 2 ) \* 1 ] + [ ( 2 ^ 1 ) \* 0 ] + [ ( 2 ^ 0 ) \* 1 ]

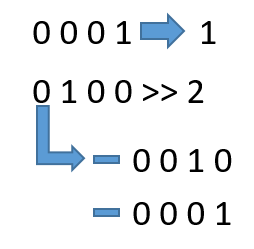
( 8 \* 0 ) + ( 4 \* 1 ) + ( 2 \* 0 ) + ( 1 \* 1 )

0 + 4 + 0 + 1 = 5









**Identify Operators:**

Used to check if the objects are the same or not.

|  |  |
| --- | --- |
| **Operator** | **Operation** |
| is | Return True if both variables are the same object |
| is not | Return True if both variables are not the same object |

1 is 1 = True

1 is “1” = False

1 is not “1” = True

**Membership Operators:**

Used to test if a sequence is present in an object.

|  |  |
| --- | --- |
| **Operator** | **Operation** |
| in | Return True if the specified value is present in the object |
| Not in | Return True if the specified value is not present in the object |

