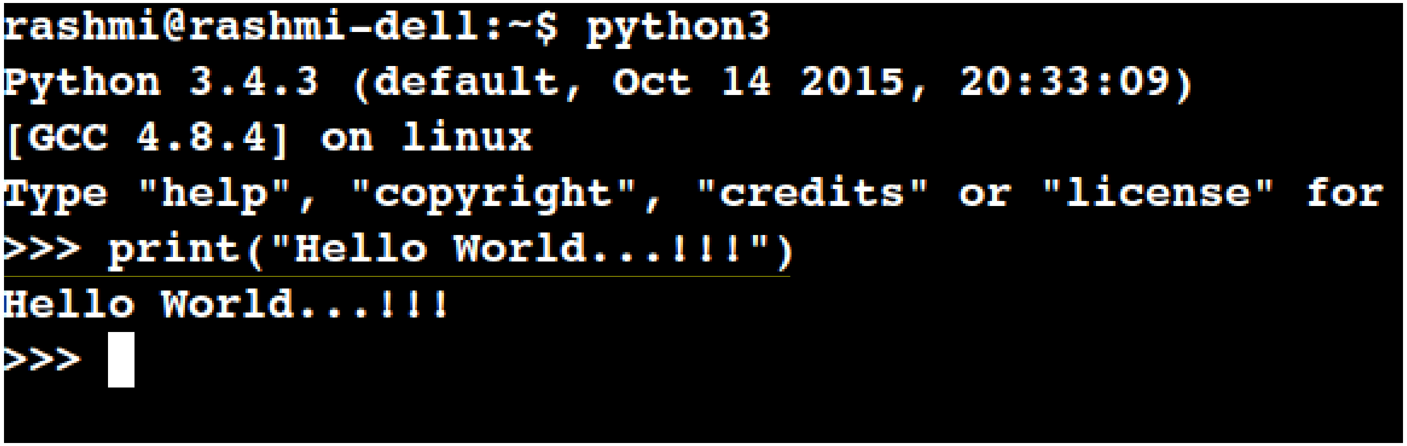
Python : Getting Started

Tushar B. Kute,
http://tusharkute.com

First Python program

- Let us execute programs in different modes of programming.
- Interactive Mode Programming:
  - Invoking the interpreter without passing a script file as a parameter brings up the following prompt:



```
rashmi@rashmi-dell:~$ python3
Python 3.4.3 (default, Oct 14 2015, 20:33:09)
[GCC 4.8.4] on linux
Type "help", "copyright", "credits" or "license" for
>>> print("Hello World...!!!")
Hello World...!!!
>>> 
```

Script Mode Programming

- Invoking the interpreter with a script parameter begins execution of the script and continues until the script is finished. When the script is finished, the interpreter is no longer active.
- Let us write a simple Python program in a script. Python files have the extension .py. Type the following source code in a test.py file-

```
print ("Hello, Python!")
```

- Now, try to run this program as follows-

```
$ python test.py
```

Python Identifiers

- A Python identifier is a name used to identify a variable, function, class, module or other object.
- An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).
- Python does not allow punctuation characters such as @, $, and % within identifiers.
- Python is a case sensitive programming language. Thus, `College` and `college` are two different identifiers in Python.

Python Identifiers – Naming Conventions

- Class names start with an uppercase letter. All other identifiers start with a lowercase letter.
- Starting an identifier with a single leading underscore indicates that the identifier is private.
- Starting an identifier with two leading underscores indicates a strong private identifier.
- If the identifier also ends with two trailing underscores, the identifier is a language-defined special name.

Keywords

- Keywords are the **reserved words** in Python.
- We cannot use a keyword as a variable name, function name or any other identifier. They are used to define the syntax and structure of the Python language.
- In Python, keywords are case sensitive.
  - All the keywords except True, False and None are in lowercase and they must be written as it is.
- There are 33 keywords in Python 3.7

Python keywords

- False class finally is return
- None continue for lambda try
- True def from nonlocal while
- and del global not with
- as elif if or yield
- assert else import pass
- break except in raise

Declaring and using variables

```
>>> num1 = 45
>>> num2 = 56
>>> print(num1)
45
>>> num3 = 12.33
>>> print(num3)
12.33
>>> name = 'Tushar'
>>> print(name)
Tushar
```

Data types

- Numbers:
  - int
  - float
  - complex
- String
- Boolean
- List
- Tuple
- Set
- Dictionary

Integers

```
>>> num = 23
>>> type(num)
<class 'int'>
>>> num + 10
33
>>> num ** 100
148861915063630393937915565865597542319
871196538013686865769882092224332785393
313521523901432773468042334765921794473
10859520222529876001
```

Integer length

- Try this:

```
>>> num ** 1000
```

- This will generate a big number with 100s of digits.
- There is NO inherent limit to the integer to store in memory. It goes on using until we run out of memory.

Floating point numbers

```
>>> num = 59.33
>>> print(num)
59.33
>>> num = 5933e18
>>> print(num)
5.933e+21
>>> type(num)
<class 'float'>
>>> num = 12.9567255478
>>> num * 11.43
148.09537301135398
```

Floating point numbers

```
>>> num1 = 4.233e221
>>> num2 = 12.322E212
>>> num1 * num
inf
>>> 2.0 ** 1023
8.98846567431158e+307
>>> 2.1 ** 1023
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  OverflowError: (34, 'Numerical result out of
range')
```

Floating point limit

Multiple assignment

```
>>> num1, num2, num3 = 12, 34, 55
>>> print(num1)
12
>>> print(num2)
34
>>> print(num3)
55
```

Multiple assignment

```
>>> num1 = num2 = num3 = 27
>>> print(num1)
27
>>> print(num2)
27
>>> print(num3)
27
```

Semicolon separator

```
>>> num1 = 12; num2 = 34; num3 = 31
>>> print(num2)
34
>>> num1 = 10; num1 = num1 + 2; print(num1)
12
```

Strings

- Strings can be declared in single or double quotes.

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

Making combinations

```
>>> data = 'Learning "Python" is fun'
>>> print(data)
Learning "Python" is fun
>>> data = "Learning 'Python' is fun"
>>> print(data)
Learning 'Python' is fun
```

String concatenation

```
>>> first = 'Python'
>>> second = 'Programming'
>>> last = first + second
>>> print(last)
PythonProgramming
>>> print(first+second)
PythonProgramming
>>> print('Python'+'Programming')
PythonProgramming
```

Escape Sequences

\n	New Line
\t	Tab
\v	Vertical tab
\r	Carriage Return
\b	Backspace
\a	Audio bell
\\	Single slash

Using escape sequences

```
>>> print('Hello\nWorld')
Hello
World
>>> print('Hello\bWorld')
HellWorld
>>> print('Hello\vWorld')
Hello
     World
>>> print('Hello\rWorld')
World
>>> print('Hello\\World')
Hello\World
```

Comment

- Python Syntax ‘Comments’ let you store tags at the right places in the code.
- You can use them to explain complex sections of code. The interpreter ignores comments.
- Declare a comment using an octothorpe / hash (#).

```
# This is a comment
>>> num = 34     #Variable declared
```

- Python does not support general multiline comments like Java or C++.

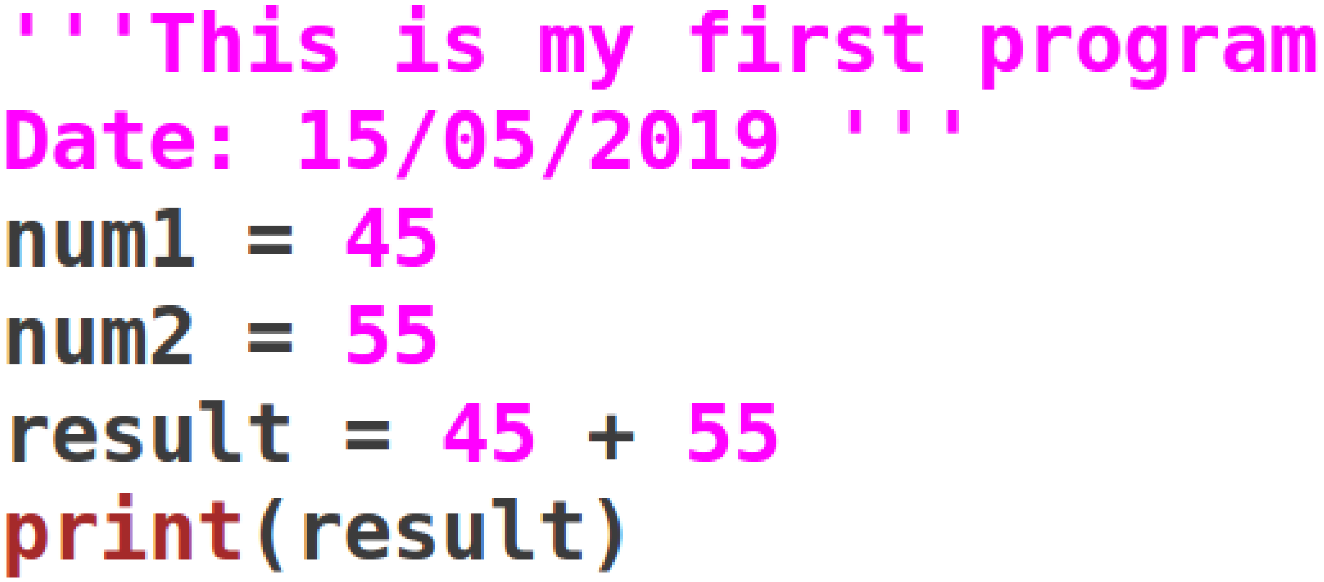
Docstring

- A docstring is a documentation string. Like a comment, this Python Syntax is used to explain code.
- But unlike comments, they are more specific. Also, they are retained at runtime.
- This way, the programmer can inspect them at runtime. Delimit a docstring using three double or single quotes.

Multi-line string

```
>>> line = '''Hello
... Welcome to MITU
... Pune'''
>>> print(line)
Hello
Welcome to MITU
Pune
```

Multi-line comment



```
'''This is my first program
Date: 15/05/2019 '''
num1 = 45
num2 = 55
result = 45 + 55
print(result)
```

Common string functions

- title()
- upper()
- lower()
- swapcase()
- isalpha()
- isdigit()
- islower()
- isupper()
- split()
- strip()
- lstrip()
- rstrip()
- startswith()
- endswith()

Using string functions

```
>>> data = 'hello'
>>> data.upper()
'HELLO'
>>> data.isalpha()
True
>>> data.split()
['hello']
>>> data.startswith('he')
True
>>> data.replace('e','a')
'hallo'
```

The dir( ) function

- The dir() function returns all properties and methods of the specified object, without the values.
- This function will return all the properties and methods, even built-in properties which are default for all object.
- If the object has __dir__() method, the method will be called and must return the list of attributes.
- If the object doesn't have __dir__() method, this method tries to find information from the __dict__ attribute (if defined), and from type object. In this case, the list returned from dir() may not be complete.

Using dir and help

- How to use dir( ) ?

  -
    ```
    >>> data = ‘hello’
    ```

  -
    ```
    >>> dir(data)
    ```
- How to see the help of functions ?

  -
    ```
    >>> help(data.upper)# Object function
    ```

  -
    ```
    >>> help(len)       # basic function
    ```
- You can apply the dir() and help() function to all kind of variables and objects.

The print function

- The print() function prints the given object to the standard output device (screen) or to the text stream file.
- The full syntax of print() is:
  - `print(*objects, sep=' ', end='\n')`
    - objects - object to the printed. * indicates that there may be more than one object
    - sep - objects are separated by sep. Default value: ' '
    - end - end is printed at last

Using print( )

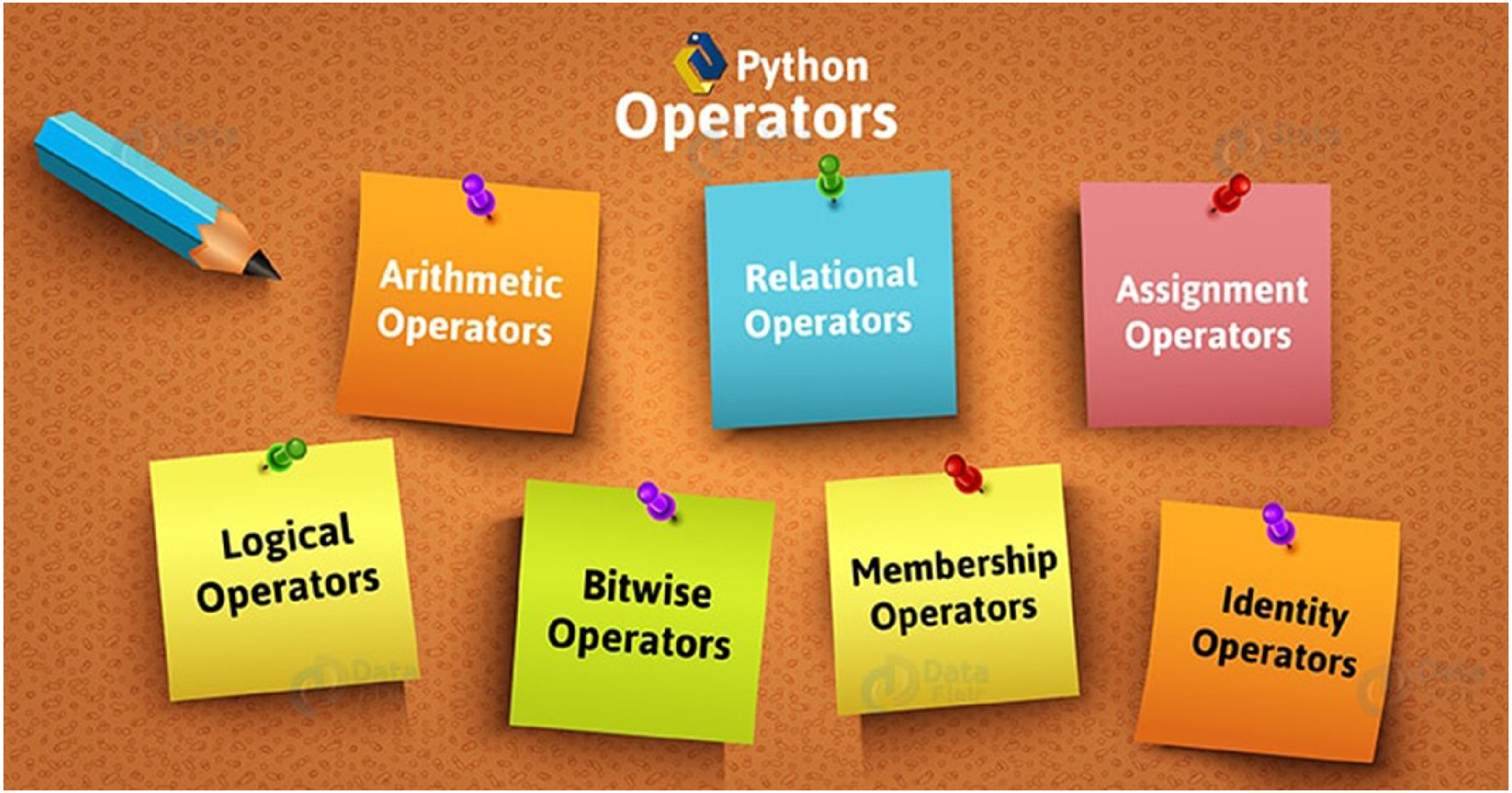
```
>>> name = 'Tushar'
>>> age = 34
>>> print('My name is',name,'and age is',age)
My name is Tushar and age is 34
>>> print('My name is %s and age is %d' %
(name,age))   # Formatted print
My name is Tushar and age is 34
>>> print('My name is {} and age is
{}' .format(name,age))  #Using .format
My name is Tushar and age is 34
```

Print options

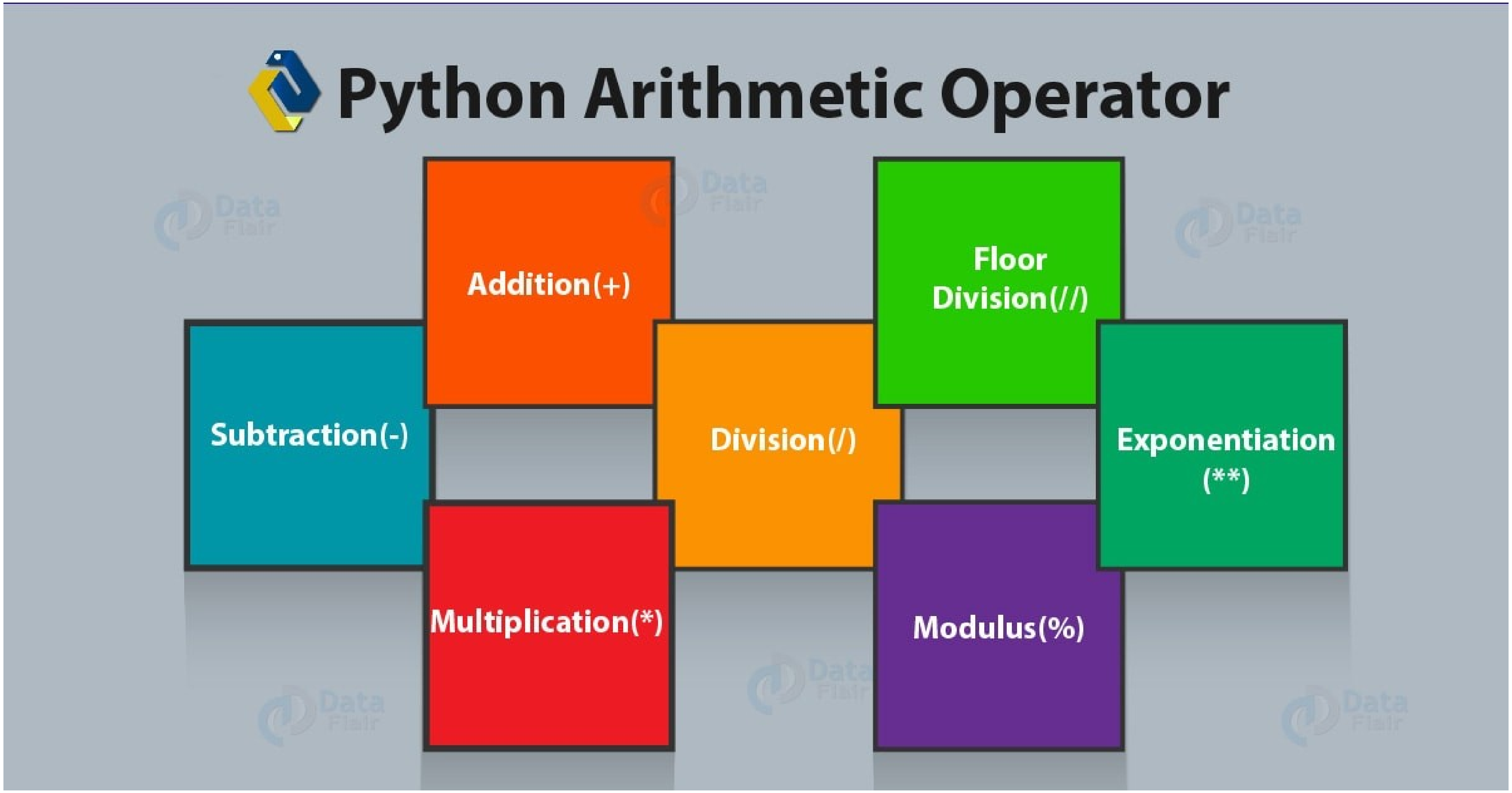
```
>>> print('My name is',name)
My name is Tushar
>>> print('My name is',name,end='\n\n')
My name is Tushar

>>> print(name,age)
Tushar 34
>>> print(name,age,sep='\t')
Tushar 34
>>> print(name,age,sep='\n')
Tushar
34
```

Operators


Python
Operators
Arithmetic Operators
Relational Operators
Assignment Operators
Logical Operators
Bitwise Operators
Membership Operators
Identity Operators

Arithmetic Operators



Arithmetic operators

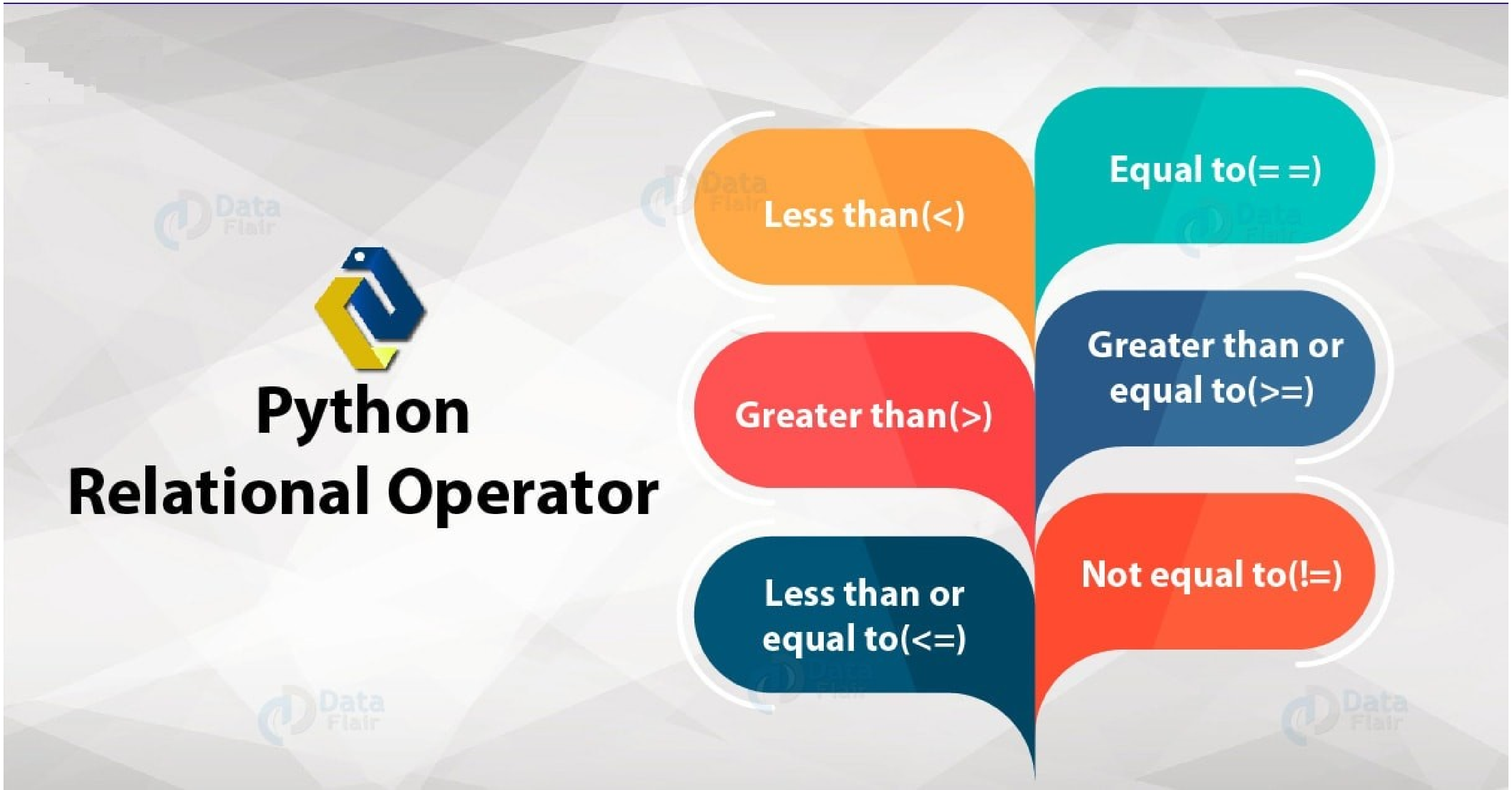
```
>>> num1 = 23; num2 = 11
>>> result = num1 + num2
>>> result
34
>>> num1 - num2
12
>>> num1 * num2
253
>>> num1 / num2
2.090909090909091
>>> num1 // num2
2
>>> num1 % num2
1
```

Arithmetic operators on strings

- The multiplication operator can be used on strings too.

```
>>> name = 'Tushar'
>>> name * 5
'TusharTusharTusharTusharTushar'
```

Relational operators



Python
Relational Operator

Less than(<)

Greater than(>)

Less than or equal to(<=)

Equal to(= =)

Greater than or equal to(>=)

Not equal to(!=)

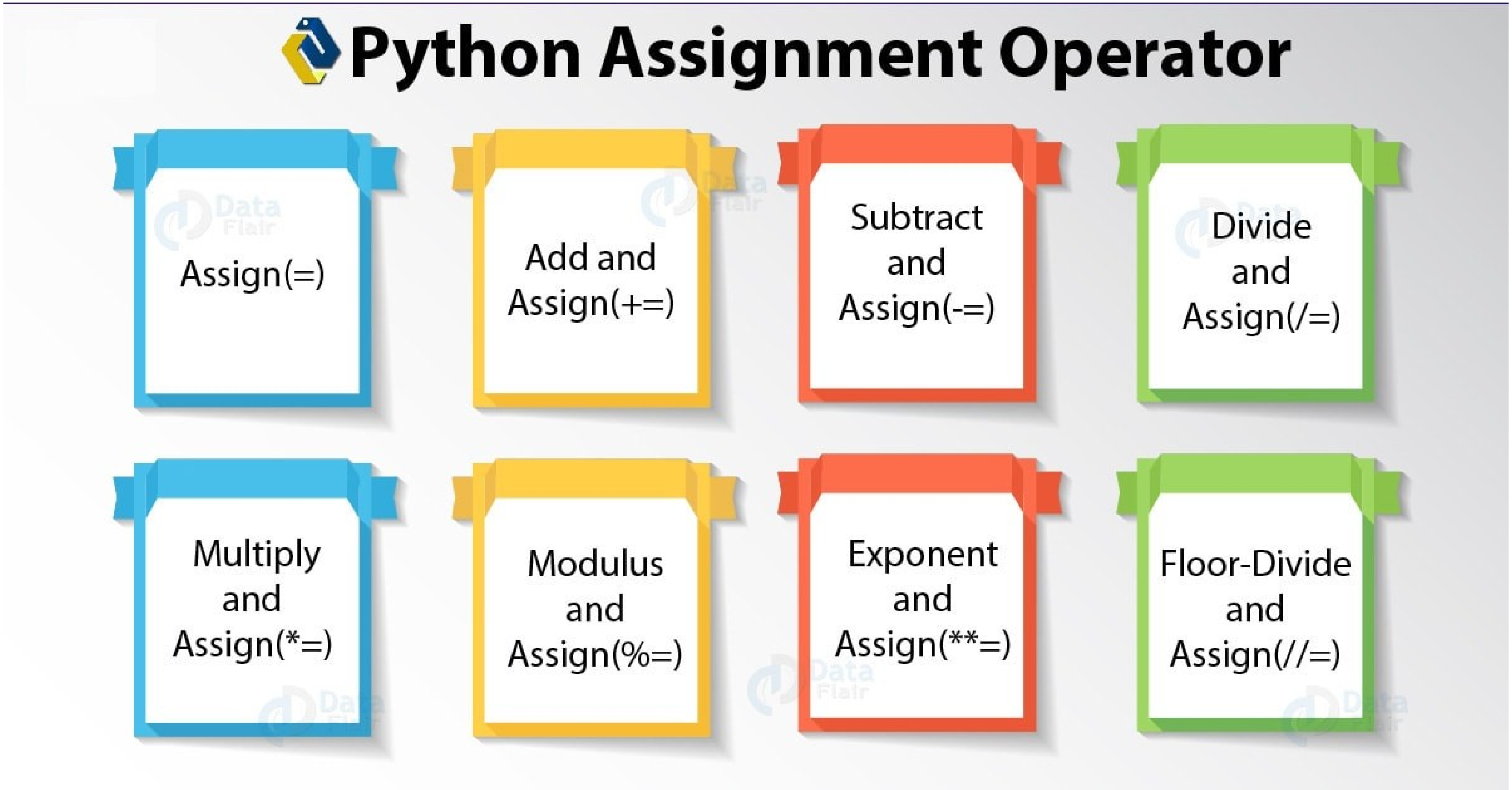
Relational operators

```
>>> num1 > num2
True
>>> num1 <= num2
False
>>> num1 == num2
False
>>> num1 != num2
True
```

Relational operators on strings

```
>>> 'Abc' != 'AbC'
True
>>> 'Abc' == 'AbC'
False
>>> 'Abc' < 'AbC'
False
>>> 'Abc' < 'AbCdef'
False
```

Assignment operators



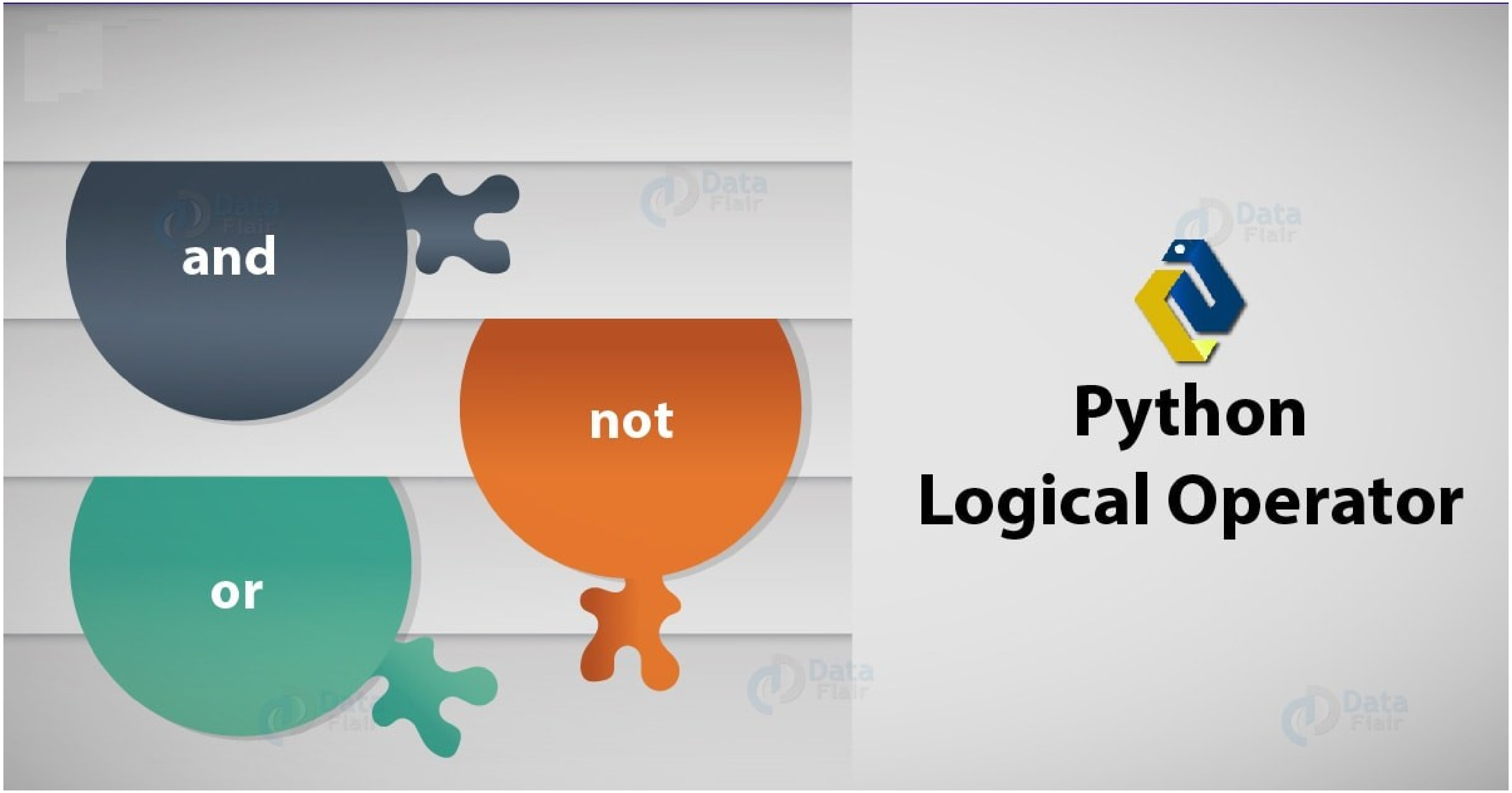
Python Assignment Operator

- Assign(=)
- Add and Assign(+=)
- Subtract and Assign(-=)
- Divide and Assign(/=)
- Multiply and Assign(*=)
- Modulus and Assign(%=)
- Exponent and Assign(**=)
- Floor-Divide and Assign(//=)

Assignment operators

```
>>> print(num1)
25
>>> num1 += 2
>>> print(num1)
27
>>> num1 *= 2
>>> print(num1)
54
>>> num1 /= 2
>>> print(num1)
27.0
```

Logical operators



Logical operators

```
>>> num1 > num2 and num1 < 100
True
>>> num1 > 100 or num1 == num2
False
>>> not num1 < 100
False
```

Membership operators

- These operators test whether a value is a member of a sequence. The sequence may be a list, a string, or a tuple. We have two membership python operators- ‘in’ and ‘not in’.
  - `in`
    - This checks if a value is a member of a sequence.
  - `not in`
    - Unlike ‘in’, ‘not in’ checks if a value is not a member of a sequence.

Membership operators

```
>>> x = 10
>>> x in [34,10,32,17]
True
>>> 15 in [34,10,32,17]
False
>>> 15 not in [34,10,32,17]
True
>>> 'kar' in 'Tendulkar'
True
```

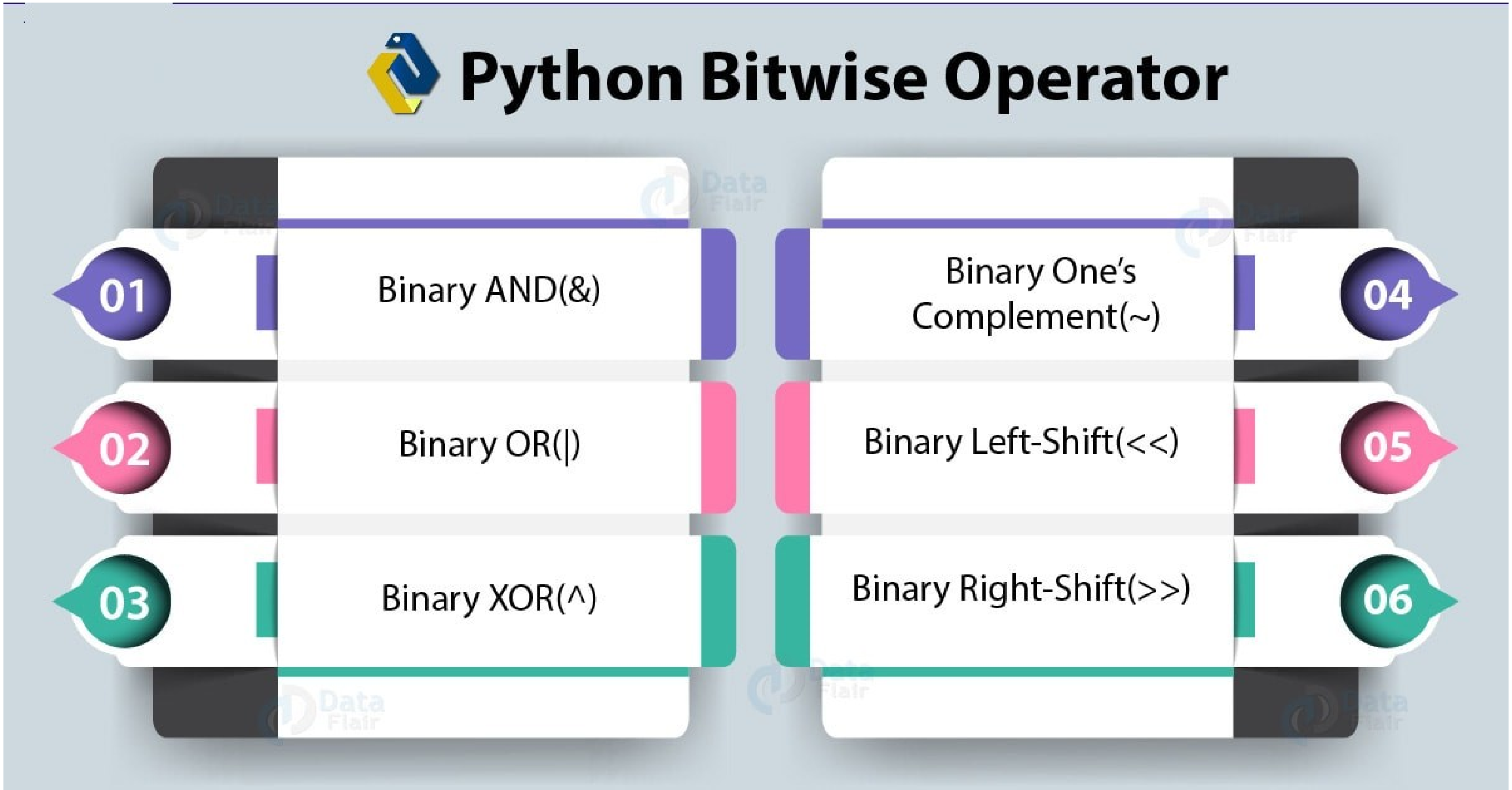
Identity operators

- These operators test if the two operands share an identity. We have two identity operators- ‘is’ and ‘is not’.
  - `is`
    - If two operands have the same identity, it returns True.
  - `is not`
    - If two operands have the different identity, it returns True.

Identity operators

```
>>> 2 is 2
True
>>> 2 is '2'
False
>>> 20 is 20.0
False
>>> 20 is not 20.0
True
>>> 2000.0 is 2e3
True
```

Bitwise operators



Python Bitwise Operator

01 Binary AND(&)

02 Binary OR(|)

03 Binary XOR(^)

04 Binary One's Complement(~)

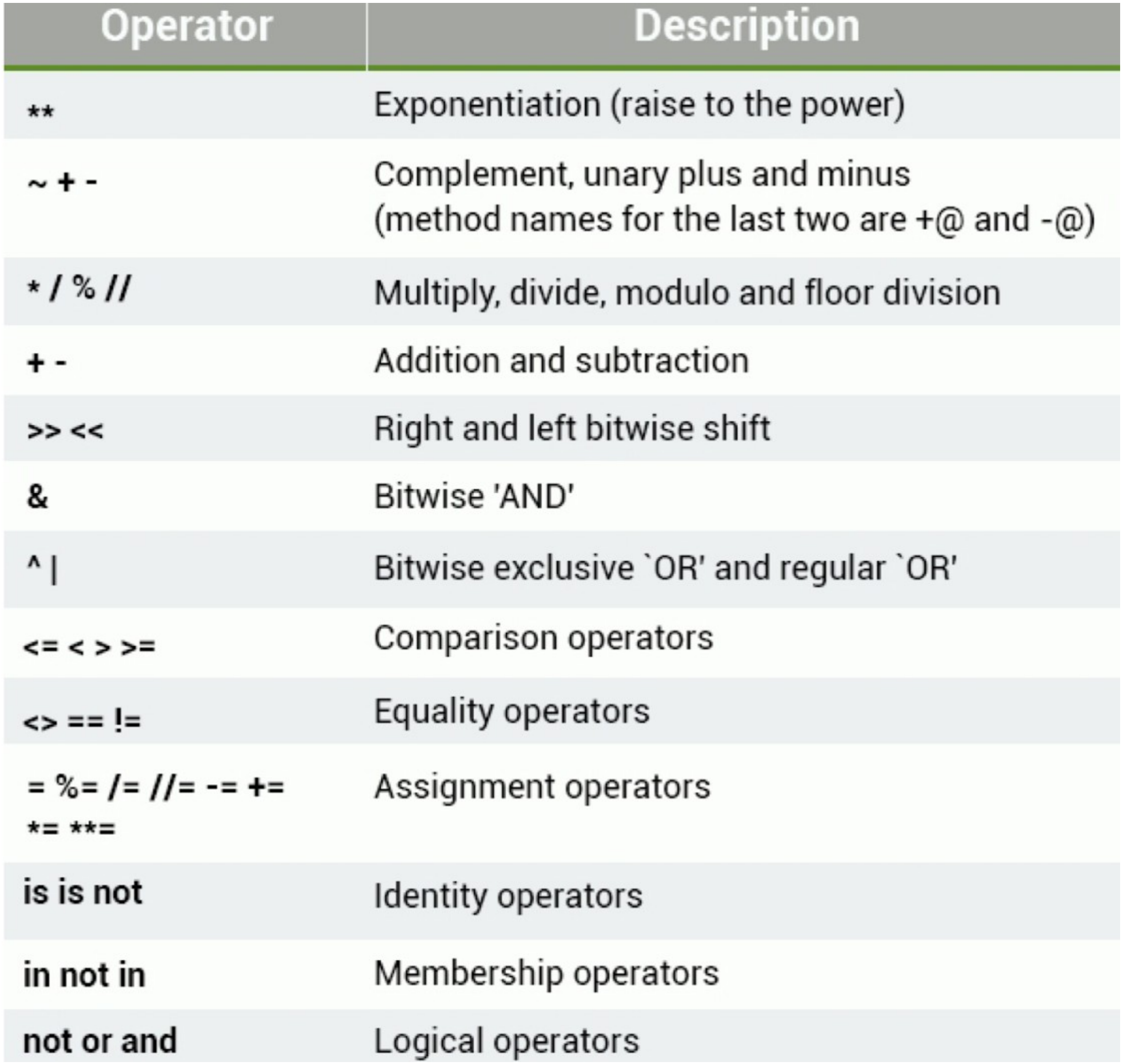
05 Binary Left-Shift(<<)

06 Binary Right-Shift(>>)

Bitwise operators

```
>>> x = 19; y = 34
>>> x & y
2
>>> x | y
51
>>> x ^ y
49
>>> y << 2
136
>>> ~x
-20
```

Operators Precedence



Operator	Description
**	Exponentiation (raise to the power)
~ + -	Complement, unary plus and minus (method names for the last two are +@ and -@)
* / % //	Multiply, divide, modulo and floor division
+ -	Addition and subtraction
>> <<	Right and left bitwise shift
&	Bitwise 'AND'
^ |	Bitwise exclusive `OR' and regular `OR'
<= < > >=	Comparison operators
<> == !=	Equality operators
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

Two more types

- Complex

```
>>> num = 2.3 + 4.5j
>>> print(num)
(2.3+4.5j)
>>> type(num)
<class 'complex'>
```

- Boolean

```
>>> num = True
>>> print(num)
True
>>> type(num)
<class 'bool'>
```

Special type: None

- The null keyword is commonly used in many programming languages, such as Java, C++, C# and Javascript. It is a value that is assigned to a variable.
- The equivalent of the null keyword in Python is **None**. It was designed this way for two reasons:
  - Many would argue that the word "null" is somewhat esoteric. It's not exactly the most friendliest word to programming novices. Also, "None" refers exactly to the intended functionality - it is *nothing*, and has no behavior
  - In most object-oriented languages, the naming of objects tend to use camel-case syntax. eg. ThisIsMyObject. As you'll see soon, Python's None type is an object, and behaves as one.

Basic use

```
>>> num = None
>>> print(num)
None
>>> num
>>> type(num)
<class 'NoneType'>
```

Type conversion

- The process of converting the value of one data type (integer, string, float, etc.) to another data type is called type conversion. Python has two types of type conversion.
  - Implicit Type Conversion
  - Explicit Type Conversion
- Implicit Type Conversion:
  - In Implicit type conversion, Python automatically converts one data type to another data type. This process doesn't need any user involvement.

Type conversion

```
>>> num1 = 45       #int
>>> num2 = 56.23    #float
>>> result = num1 + num2
>>> print(result)   #float
101.22999999999999
```

Explicit type conversion

- In Explicit Type Conversion, users convert the data type of an object to required data type. We use the predefined functions like int(), float(), str(), etc to perform explicit type conversion.
- This type conversion is also called typecasting because the user casts (change) the data type of the objects.
- Syntax :

```
(required_datatype) (expression)
```

- Typecasting can be done by assigning the required data type function to the expression.

Explicit type conversion

```
>>> num1 = 45
>>> num2 = 56.23
>>> result = num1 + int(num2)
>>> print(result)
101
```

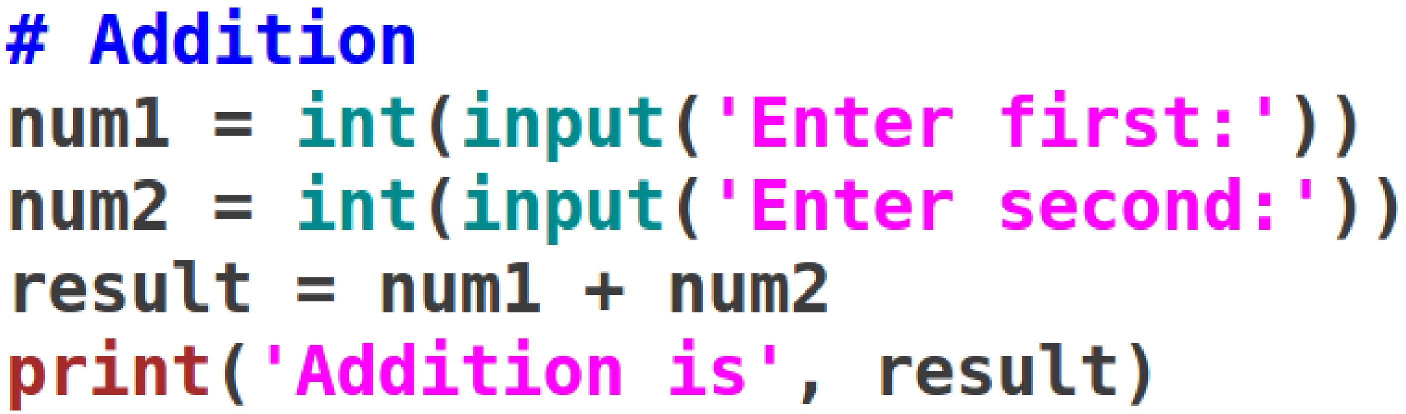
Type casting on strings

```
>>> num = '178'
>>> num * 3
'178178178'
>>> int(num) * 3
534
>>> num = 123
>>> s = 'hello' + str(num)
>>> s
'hello123'
```

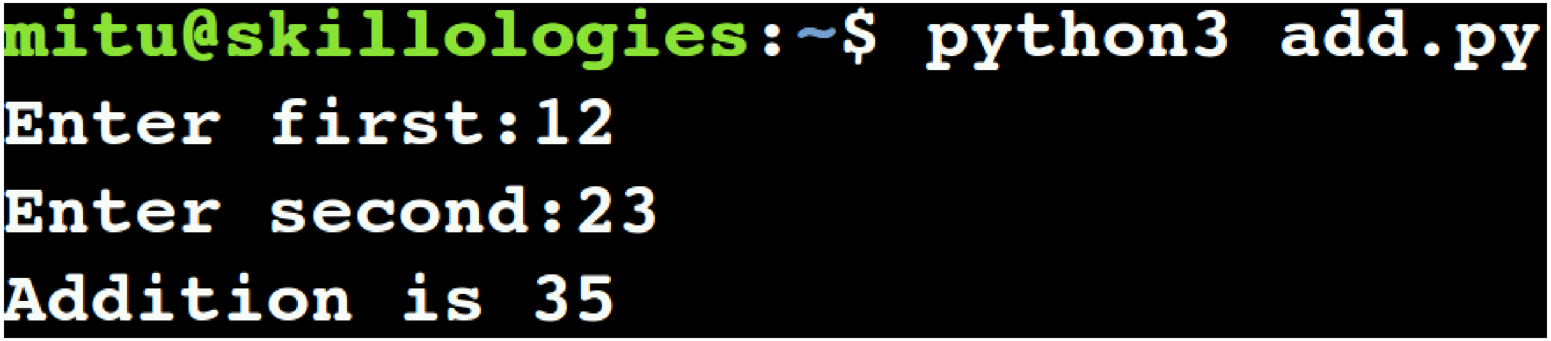
Taking user input

- The input( ) function is used to read the values from keyboard. It prints the string and reads a string from keyboard which then will be stored in a variable.
- Example:
  - `s = input('Enter your name:')`
  - `num = int(input('Enter a number:'))`
  - `marks = float(input('Enter marks:'))`

Sample code:



```python
# Addition
num1 = int(input('Enter first:'))
num2 = int(input('Enter second:'))
result = num1 + num2
print('Addition is', result)
```



```
mitu@skillologies:~$ python3 add.py
Enter first:12
Enter second:23
Addition is 35
```

Exercises

- Write a program to read Celsius temperature and print equivalent Fahrenheit temperate on screen.
- Read radius of the circle from user and find the area and perimeter of it.
- Read the amount and percentage of interest from the keyboard and find final amount after adding interest in original amount.
- Write a program to read distance value in meters and convert it into centimeters, inches, and yards.

Thank you

@mitu_skillologies

/mITuSkillologies

@mitu_group

/company/mitu-skillologies

c/MITUSkillologies

Web Resources
http://mitu.co.in
http://tusharkute.com

contact@mitu.co.in

tushar@tusharkute.com