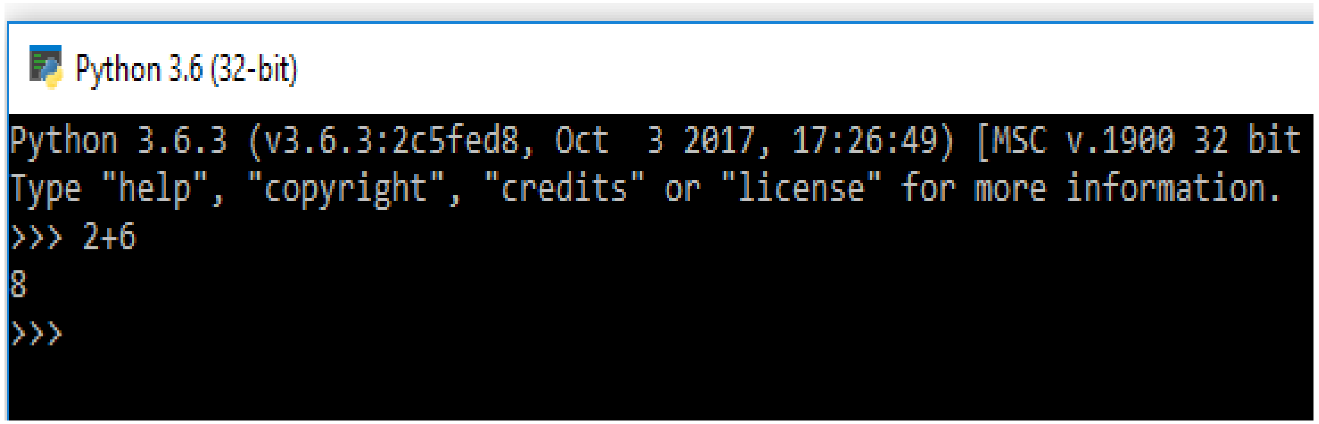
Interpreting a Python Code

Python is an interpreted programming language. Python source code is compiled to bytecode as a **.pyc** file, and this bytecode can be interpreted.

There are two modes for using the Python interpreter:

- Interactive Mode - Without passing the python script file to the interpreter, directly execute code to Python prompt.



```
Python 3.6.3 (v3.6.3:2c5fed8, Oct  3 2017, 17:26:49) [MSC v.1900 32 bit
Type "help", "copyright", "credits" or "license" for more information.
>>> 2+6
8
>>>
```

- Script Mode - programmers can store Python script source code in a file with the .py extension, and use the interpreter to execute the contents of the file. For example, you might have to type on the shell as:

>>> python myfile.py

Inserting Comments in a Program

Comments are non-executable statements in Python. It means neither the python compiler nor the PVM will execute them.

- Single Line Comments: A single-line comment begins with a hash (#) symbol.

```
#Defining a variable to store number.
n = 50 #Store 50 as value into variable n.
```

- Multi-line comments: Triple double quote (""") and single quote (''') are used for Multi-line commenting. Also called block comments.

```
'''
Author: www.w3schools.in
Description:
Writes the words Hello World on the screen
'''
```

Indentation

- Indentation in Python refers to the (spaces and tabs) that are used at the beginning of a statement. The statements with the same indentation belong to the same group called a **suite**.
- Python uses indentation to indicate a **block of code**.

Example:

```
if 5 > 2:
  print("Five is greater than two!")
```

Syntax Error:

```
if 5 > 2:
print("Five is greater than two!")
```

The number of spaces is up to you as a programmer, but it has to be at least one.

Example

```
if 5 > 2:
 print("Five is greater than two!")
if 5 > 2:
        print("Five is greater than two!")
```

Understanding the Basic Syntax

Variables

Python has no command for declaring a variable.

A variable is created the moment you first assign a value to it.

```
x = 15

X=1
y = “Ravi“
```

Casting

If you want to specify the data type of a variable, this can be done with casting.

```
x = str(3)    # x will be '3'
y = int(x)    # y will be 3
z = float(3)  # z will be 3.0
```

You can get the data type of a variable with the **type()** function.

```
y = "John"
print(type(y))  #<class 'str'>
```

Variable is a name that is used to refer to memory location. Python variable is also known as an identifier and used to hold value.

- Python smart enough to get variable type.
- There are few rules to follow while naming the Python Variable.
- A variable name must start with either an **English letter or underscore (_).**
- A variable name **cannot** start with the **number**.
- Variable name must **not** contain any **white-space**, or **special character (!, @, #, %, ^, &, *).**
- The variable's name **is case sensitive**.
- It is **recommended** to **use lowercase letters** for the variable name. Rahul and rahul both are two different variables.

Find wrong ones

1. name = "A"
2. Name = "B"
3. naMe = "C"
4. 6NAME = "D"
5. n_a_m_e@#$*&^! = "E"
6. _na me = "F"
7. name_ = "G"
8. _name_ = "H"

Find wrong ones

1. name = "A"
2. Name = "B"
3. naMe = "C"
4. 6NAME = "D"
5. n_a_m_e@#$*&^! = "E"
6. _na me = "F"
7. name_ = "G"
8. _name_ = "H"

- Variables can hold values, and every value has a data-type. Python is a dynamically typed language; hence we do not need to define the type of the variable while declaring it. The interpreter implicitly binds the value with its type.
- Python provides us the **type()** function, which returns the type of the variable passed.
- Python provides various standard data types that define the storage method on each of them
-

Data Types

Variables can store data of different types.

Python has the following data types built-in by default:

Text Type:	str
Numeric Types:	int, float, complex
Sequence Types:	list, tuple, range
Mapping Type:	dict
Set Types:	set, frozenset
Boolean Type:	bool
Binary Types:	bytes, bytearray, memoryview

Numbers

Python supports four different numerical types –

- **int** (signed integers)
- **long** (long integers, they can also be represented in octal and hexadecimal)
- **float** (floating point real values)
- **complex** (complex numbers)

Long	float	Complex
51924361L	15.20	3.14j
0x19323L	-21.9	45.j
-052318172735L	32.3+e18	3+26j

Boolean Values

- The bool() function allows you to evaluate any value, and give you True or False in return

```
print(bool("Hello"))
print(bool(1))
```

- When you compare two values, the expression is evaluated and Python returns the Boolean answer

```
print(10 > 9)
print(10 == 9)
print(10 < 9)
```

Operators in Python

Types of Python Operators:

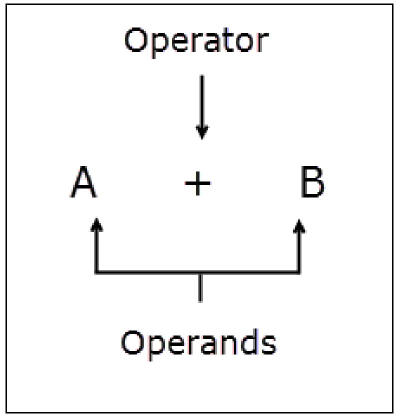
1. Arithmetic Operators
2. Assignment Operators
3. Comparison (Relational) Operators
4. Logical Operators
5. Identity Operators
6. Bitwise Operators
7. Membership Operators

Operators and Operands

- Python operators are symbols that are used to perform mathematical or logical manipulations.
- Operands are the values or variables with which the operator is applied to, and values of operands can manipulate by using the operators.

Let us take a Scenario:

6 + 2=8, where there are two operands and a plus (+) operator, and the result turns 8.



Arithmetic Operators

Symbol	Operator Name	Description
+	Addition	Adds the values on either side of the operator and calculate a result.
-	Subtraction	Subtracts values of right side operand from left side operand.
*	Multiplication	Multiplies the values on both sides of the operator.
/	Division	Divides left side operand with right side operand.
%	Modulus	It returns the remainder by dividing the left side operand with right side operand
**	Exponent	Calculates the exponential power
//	Floor Division	Here the result is the quotient in which the digits after decimal points are not taken into account.

Assignment Operators

Symbol	Operator Name	Description
=	Equal	Assigns the values of the right side operand to the left side operand.
+=	Add AND	Adds right-side operand value to the left side operand value and assigns the results to the left operand.
-=	Subtract AND	Subtracts right-side operand value to the left side operand value and assigns the results to the left operand.
*=	Multiply AND	Similarly does their respective operations and assigns the operator value to the left operand.
/=	Division AND	
%=	Modulus AND	
**=	Exponent AND	
//=	Floor Division AND	

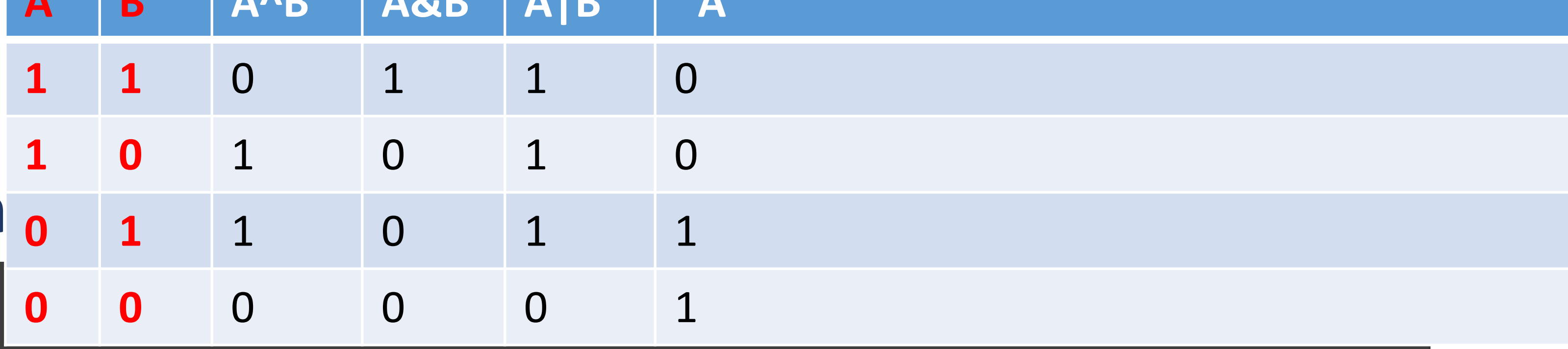
Comparison (Relational) Operators

Symbol	Operator Name	Description
==	Double Equal	If the two value of its operands are equal, then the condition becomes true, otherwise false
!= or <>	Not Equal To	If two operand's values are not equal, then the condition becomes true. Both the operators define the same meaning and function
>	Greater Than	If the value of the left-hand operand is greater than the value of right-hand operand, the condition becomes true.
<	Less Than	If the value of the left-hand operand is less than the value of the right operand, then the condition becomes true.
<=	Less Than Equal To	If the value of the left-hand operand is less than or equal to the value of right-hand operand, the condition becomes true.
>=	Greater Than Equal To	If the value of the left-hand operand is greater than or equal to the value of right-hand operand, the condition becomes true.

Logical Operators

Symbol	Operator Name	Description
or	Logical OR	If any of the two operands are non-zero, then the condition is true. A B result(A OR B) A AND B T T T T T F T F F T T F F F F F
and	Logical AND	If both the operands are true, then the condition is true.
not	Logical NOT	It is used to reverse the logical state of its operand.

Bitwise Operators



A	B	A^B	A&B	A|B	A
1	1	0	1	1	0
1	0	1	0	1	0
0	1	1	0	1	1
0	0	0	0	0	1

These operators are used to m

Symbol	Operator Name	
&	Binary AND	This operator copies the bit to the result if it exists in both operands.
|	Binary OR	This operator copies the bit if it exists in either of the operands.
^	Binary XOR	This operator copies the bit if it is set in one operand but not both.
~	Binary 1s Complement	This is a unary operator and has the ability of 'flipping' bits.
<<	Binary Left Shift	The left operands value is moved left by the number of bits specified by the right operand using this operator. PRINT(5<<1)=0101<<1=01010=1*8+1*2=10
>>	Binary Right Shift	The left operands value is moved right by the number of bits specified by the right operand using this operator.

Membership Operators

Symbol	Operator Name	Description
in	in	The result of this operation becomes True if it finds a value in a specified sequence & False otherwise.
not in	not in	result of this operation becomes True if it doesn't find a value in a specified sequence and False otherwise.

In Python, we can check whether a string or character is a member of another string or not using "**in**" or "**not in**" operators. While comparing the string, these operators consider uppercase and lowercase letters or strings separately and make case sensitive comparisons.

Example:

```
str1 = input('Please enter first string: ')
str2 = input('Please enter second string: ')
if str2 in str1:
    print(str2+' found in the first string.')
else:
    print(str2+' not found in the first string.')
```

Identity Operators

These operators are used to determine whether a value is of a certain class or type. They are usually used to determine the type of data a certain variable contains. There are two types of identity operators. These are:

Symbol	Operator Name	Description
is	is	The result becomes true if values on either side of the operator point to the same object and False otherwise.
is not	is not	The result becomes False if the variables on either side of the operator point to the same object

Identity Operators

Example:

```
# Python program to illustrate the use
# of 'is' identity operator
x = 5
if (type(x) is int):
    print("true")
else:
    print("false")
```

Example:

```
# Python program to illustrate the
# use of 'is not' identity operator
x = 5.2
if (type(x) is not int):
    print("true")
else:
    print("false")
```

Difference between == and is operators:

The Equality operator (==) compares the values of both the operands and checks for value equality. On the other hand, ‘**is**’ operator checks whether both of the operands refer to the same class or not.

Ternary Operator

Ternary operators also known as conditional expressions are operators that evaluate something based on a condition being true or false. It simply allows to test a condition in a single line replacing the multiline if-else making the code compact.

Syntax :

[on_true] **if** [expression] **else** [on_false]

Example:

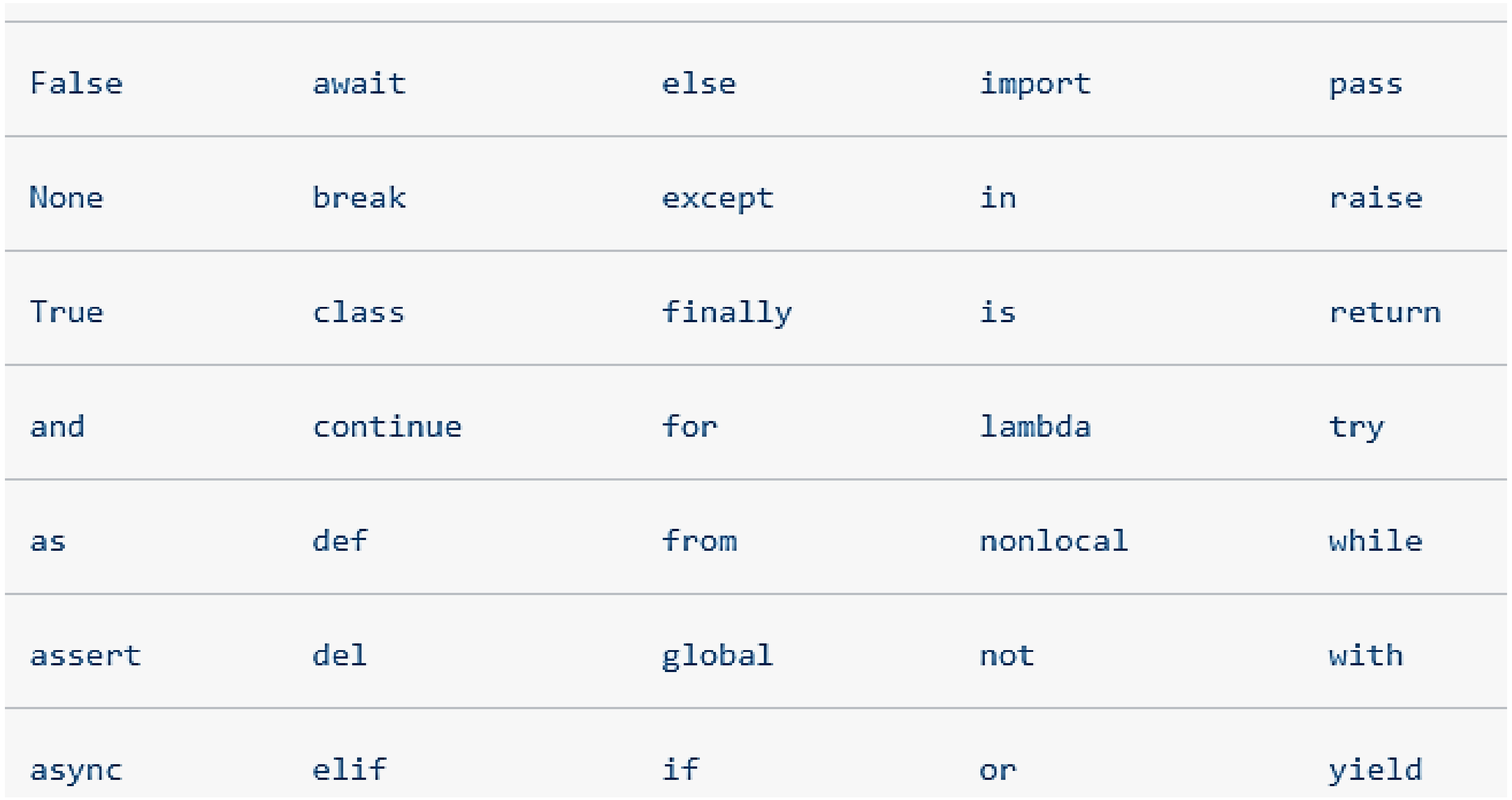
```
# Program to demonstrate conditional operator
a, b = 10, 20

# Copy value of a in min if a < b else copy b
min = a if a < b else b

print(min)
```

Output:

```
10
```

Python Keywords

- Python keywords are special reserved words that have specific meanings and purposes and can’t be used for anything but those specific purposes. As of Python 3.8, there are **thirty-five** keywords in Python.



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

- You can get a list of available keywords by using help():

```
>>>
help("keywords")
```