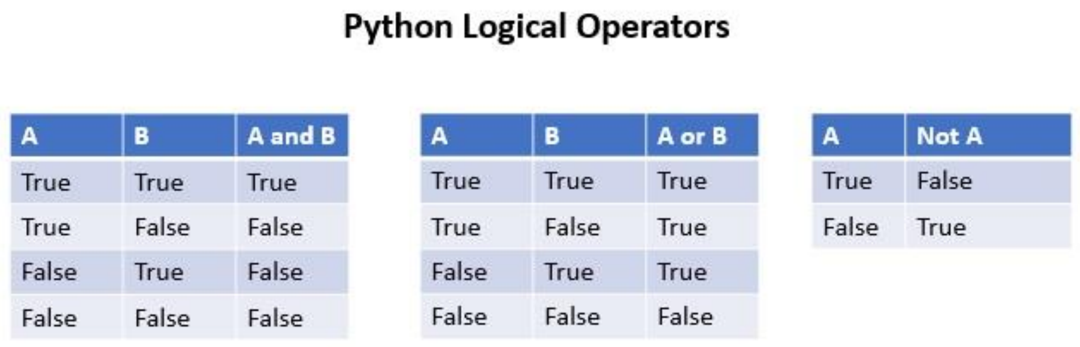
LECTURE SIX

Logical operators

Logical operators in python: **and**, **or**, **not** operators. Logical operators are used to combine conditional statements:



Python Logical Operators

A	B	A and B
True	True	True
True	False	False
False	True	False
False	False	False

A	B	A or B
True	True	True
True	False	True
False	True	True
False	False	False

A	Not A
True	False
False	True

Logical Operator	Meaning	Example
and	True if both the operands are true	x and y
or	True if either of the operands is true	x or y
not	True if operand is false (complements the operand)	not x

Example: Logical operators

```
a,b = 5,6
print((a > 2) and (b >= 6))
print(True and False)
print(True or False)
print(not True)
print (not(x < 5 and x < 10))
```

Output

```
True
False
True
 False
True
```

Bitwise operators

bitwise operators are used to perform bitwise calculations on integers. The integers are first converted into binary and then operations are performed on each bit or corresponding pair of bits, hence the name bitwise operators. The result is then returned in decimal format.

Operator	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR / Bitwise XOR
~	Bitwise inversion (one's complement)
<<	Shifts the bits to left / Bitwise Left Shift
>>	Shifts the bits to right / Bitwise Right Shift

In the table below: Let x = 10 (1010 in binary) and y = 4 (0100 in binary)

Example: Bitwise operators

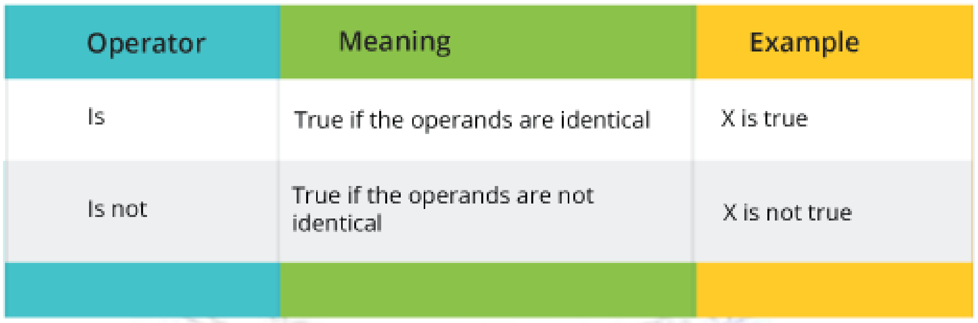
```
a,b = 10,4
print("a & b =", a & b)
print("a | b =", a | b)
print("~a =", ~a)
print("a ^ b =", a ^ b)
print(' a>> 2  =', a>> 2)
print(' a<< 1 =', a<< 1)
```

Output

```
a & b = 0
a | b = 14
~a = -11
a ^ b = 14
a>> 2  =2
a<< 1 =20
```

Identity operators

is and **is not** are the identity operators in Python. They are used to check if two values (or variables) are located on the same part of the memory. Two variables that are equal does not imply that they are identical.



Operator	Meaning	Example
Is	True if the operands are identical	X is true
Is not	True if the operands are not identical	X is not true

Example: Identity operators

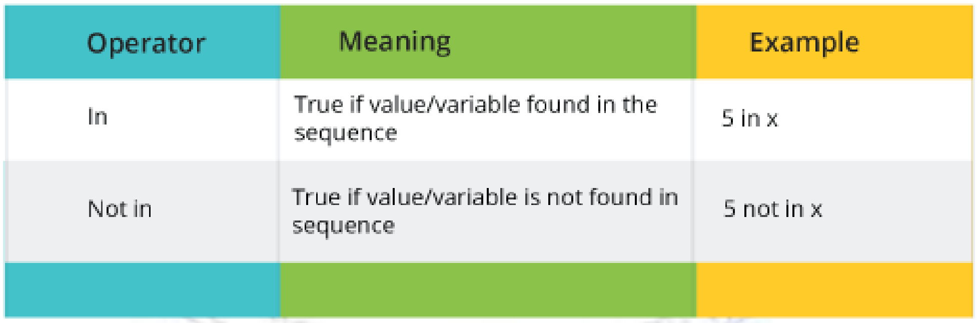
```
a,b = 5,5
x2 = 'Hello'
y2 = x2
print(a is not b)
print(x2 is y2)
```

Output

```
False
True
```

Membership operators

in and **not** in are the membership operators in Python. Python offers two membership operators to check or validate the membership of a value. is used to check if a character/ substring/ element exists in a sequence or not.



Operator	Meaning	Example
In	True if value/variable found in the sequence	5 in x
Not in	True if value/variable is not found in sequence	5 not in x

Example: Membership operators

```
x = 'Hello world'
v=[1,2,3,4,5]
m=8
print('H' in x)
print('ll' in x)
print('eo' not in x)
print(m in v)
```

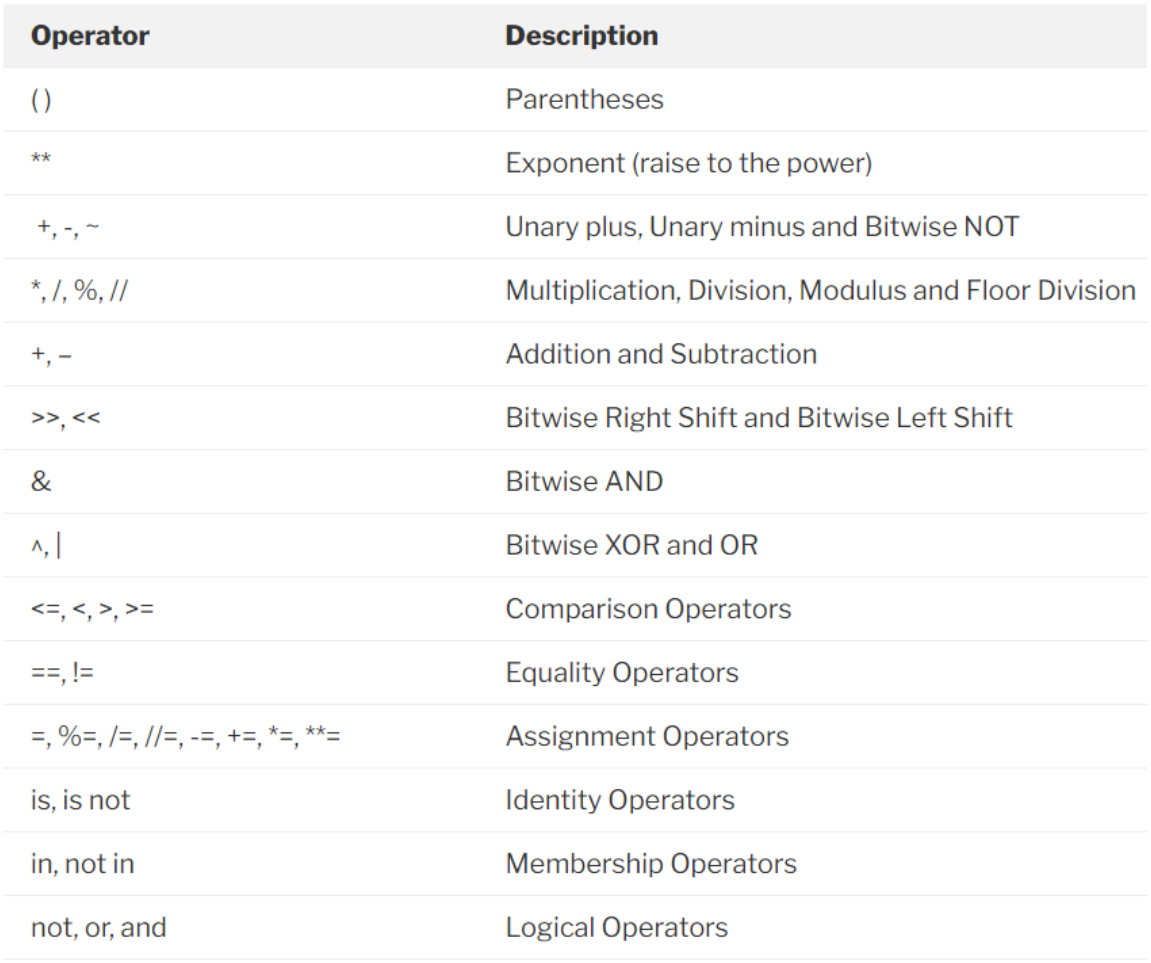
Output

```
True
True
True
False
```

Precedence of Python Operators

The precedence of an operator specifies how "tightly" it binds two expressions together.

Here is the list of Python Operators in descending order, listing from higher precedence to lower precedence.



Operator	Description
( )	Parentheses
**	Exponent (raise to the power)
+, -, ~	Unary plus, Unary minus and Bitwise NOT
*, /, %, //	Multiplication, Division, Modulus and Floor Division
+, –	Addition and Subtraction
>>, <<	Bitwise Right Shift and Bitwise Left Shift
&	Bitwise AND
^, |	Bitwise XOR and OR
<=, <, >, >=	Comparison Operators
==, !=	Equality Operators
=, %=, /=, //=, -=, +=, *=, **=	Assignment Operators
is, is not	Identity Operators
in, not in	Membership Operators
not, or, and	Logical Operators

Example: Precedence of Python Operator

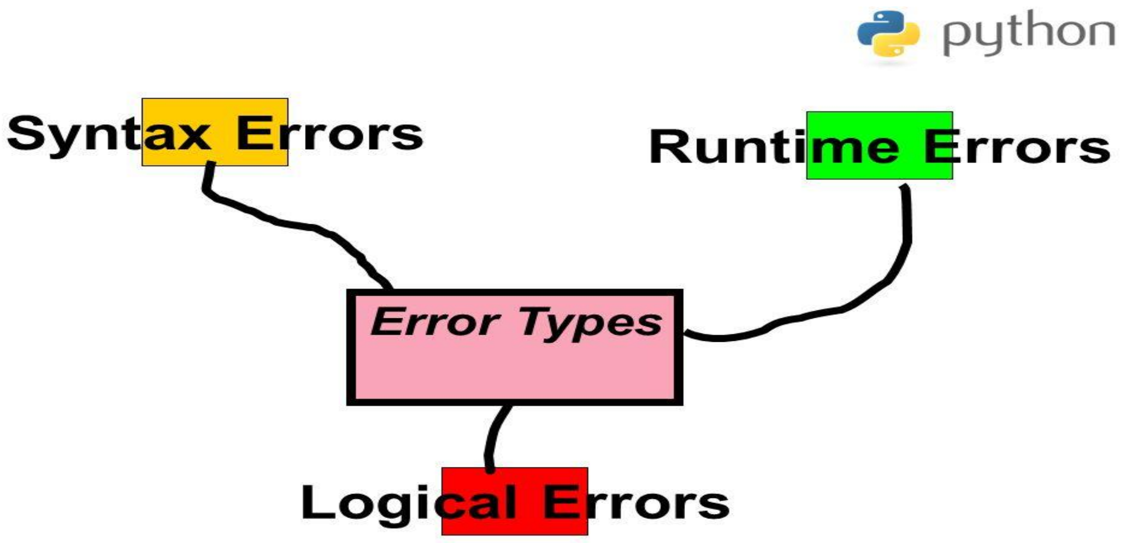
a,b,c,d,e = 20,10,15,5,0
e = (a + b) * c / d
print ("e= ", e)
e = ((a + b) * c) / d
print ("e= ", e)
e = (a + b) * (c / d)
print ("e= ", e)
e = a + (b * c) // d;
print ("e= ", e)
m= 10 - 4 * 2
print ("m= ", m)
m= (10 - 4) * 2
print ("m= ", m)
print(2 ** 3 ** 2)
print((2 ** 3) ** 2)

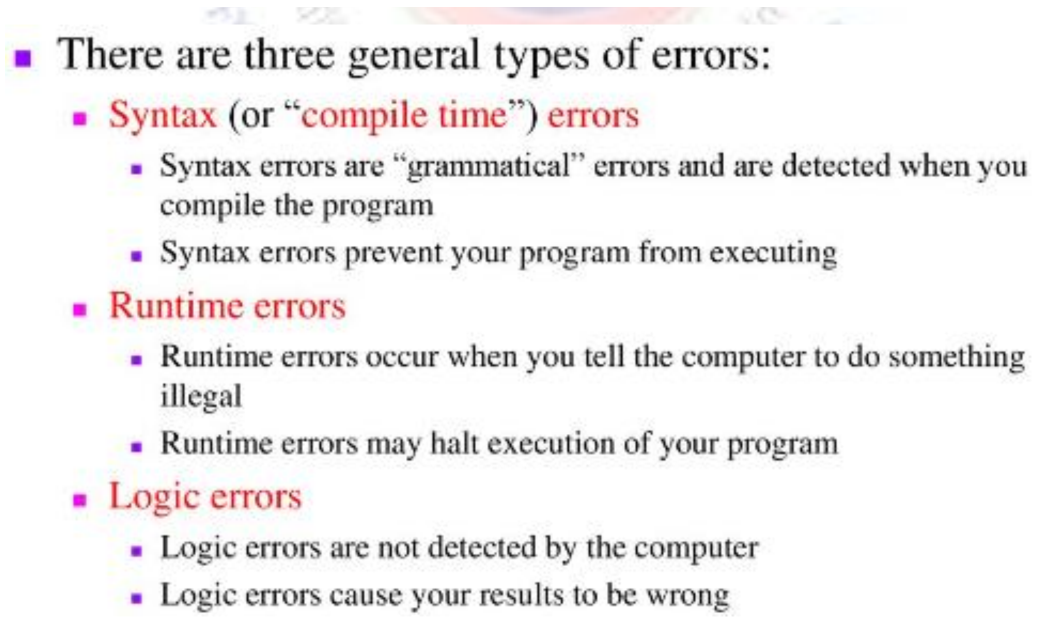
Output

```
e=  90.0
e=  90.0
e=  90.0
e=  50
m=  2
m=  12
512
64
```

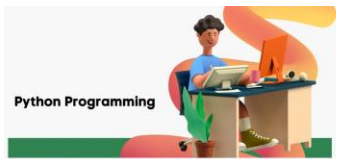
Different Types of Python Errors

- **Beginning programmers** make **mistakes** writing programs because of **inexperience** in programming in general or due to **unfamiliarity** with a programming language.
- **Seasoned programmers** make **mistakes** due to **carelessness** or because the proposed solution to a problem is **faulty**.





- There are three general types of errors:
  - Syntax (or "compile time") errors
    - Syntax errors are "grammatical" errors and are detected when you compile the program
    - Syntax errors prevent your program from executing
  - Runtime errors
    - Runtime errors occur when you tell the computer to do something illegal
    - Runtime errors may halt execution of your program
  - Logic errors
    - Logic errors are not detected by the computer
    - Logic errors cause your results to be wrong



Syntax Errors

A syntax error is a common error that the interpreter can detect during the **translation phase** when attempting to translate a Python statement into machine language.

For example:

- y + 2 = x

  SyntaxError: can't assign to operator
- x =)3 + 4)

  SyntaxError: invalid syntax
- x = 'hello"

  SyntaxError: EOL while scanning string literal
- x = 2

  ```
  x = 2
      y = 5
  ```

  IndentationError: unexpected indent

Run-time Exceptions

Run-time exceptions arise during the **execution phase** after the interpreter's translation phase.

.

- x = y + 2

  NameError: name 'y' is not defined
- x = 5 / 0

  ZeroDivisionError: division by zero
- x = "Coronavirus" / 0

  TypeError: unsupported operand type(s) for /: 'str' and 'int'

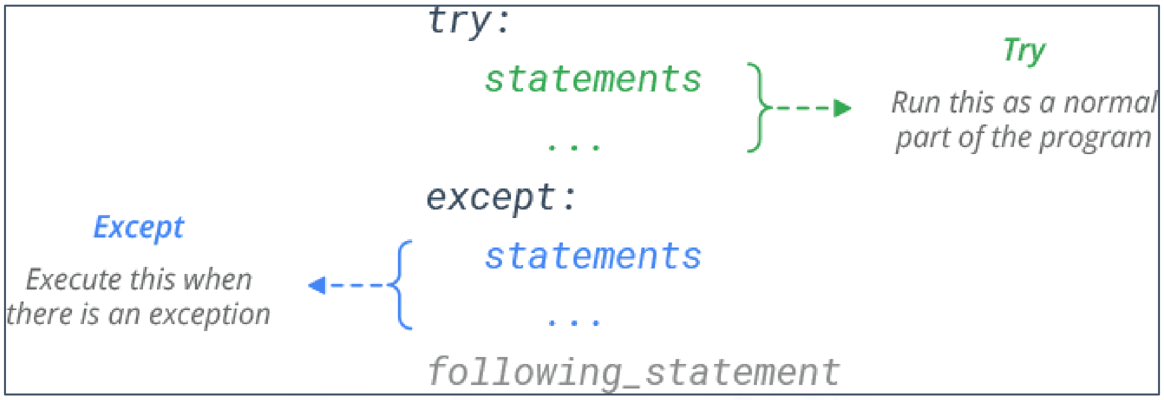
Logic Errors

happen when your code does what you told it to do, but not what you wanted it to do, like adding two numbers instead of subtracting them.Logical errors cause the program to behave incorrectly, but they do not usually crash the program.

- z = x+y /2
- z = (x+y) /2

Handling Exceptions with Try/Except

We can handle errors by the Try/Except method. we write unsafe code in the try, fall back code in except.


try:
statements
...
Try
Run this as a normal part of the program
except:
Except
Execute this when there is an exception
statements
...
following_statement

Example: Handling Exceptions with Try/Except

```
try:
    a = 10
    b = 0
    result = a/b
    print(result)
except:
    print("Error: b cannot be 0.")
```

Output

```
ERROR!
Error: b cannot be 0.
```

Example: Handling Exceptions with Try/Except

```
try:
    a = 10
    b = 0
    result = a/b
    print(result)
except ValueError as e:
     print(e)
```

Output

```
ERROR!
ZeroDivisionError: division by zero
```