



SNS COLLEGE OF ENGINEERING



Kurumbapalayam(Po), Coimbatore – 641 919

Accredited by NAAC-UGC with 'A' Grade

Approved by AICTE, Recognized by UGC & Affiliated to Anna University, Chennai

Department of CSE-IOT



Why we are here now?????



What we going to do???



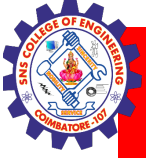
The Answer is:

Explore Python basics



What is Python?





What is Programming language? ???

A program is a set of instructions that help computer to perform tasks.

The languages that are used to write a program or set of instructions are called "Programming languages"

Programming languages are broadly categorized into three types –

1. Machine level language
2. Assembly level language
3. High-level language



Python

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

Python is an open-source, high-level, dynamically-typed, portable, expressive, easy to learn, and code programming language.

It is used for:

- web development (server-side),
- software development,
- mathematics,
- system scripting.



Why Python?

- Python works on **different platforms** (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a **simple syntax** similar to the English language.
- Python has syntax that allows developers to write programs with **fewer lines** than some other programming languages.
- Python runs on an **interpreter** system, meaning that code can be executed as soon as it is written.



Where to execute python??





Here are the ways with which we can run a Python script.

- Interactive Mode
- Command Line
- Text Editor (VS Code)
- IDE (PyCharm)





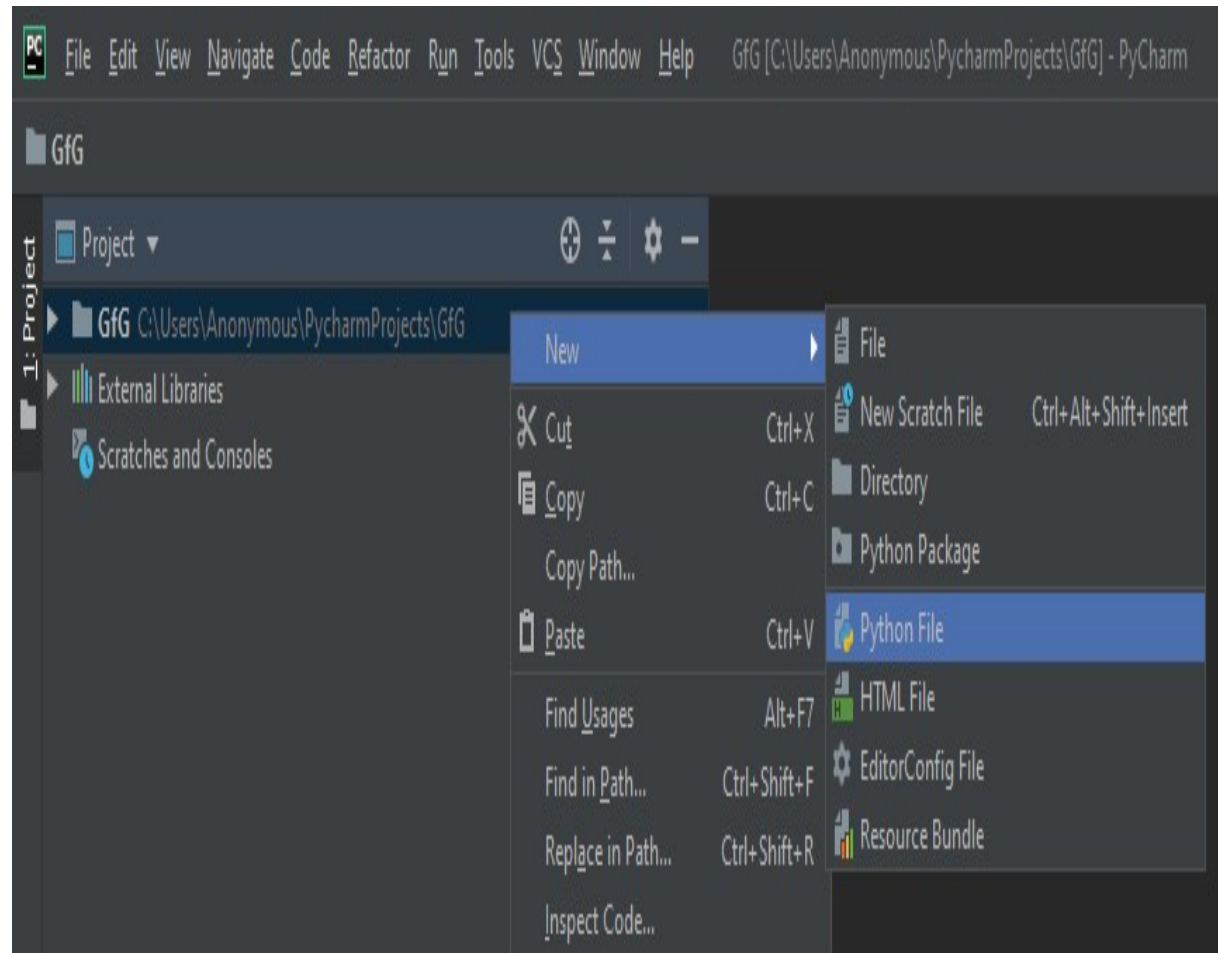
Explanation

- Interactive Mode: Command Prompt on your windows machine
- Command Line-store in a '.py' file in command line, we have to write 'python' keyword before the file name in the command prompt.
- IDE (PyCharm)



Where we going to execute??

IDE (PyCharm)





IDE

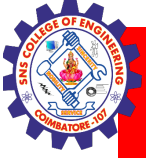


```
hello.py x
1 print('Hello World')
2
```

- Show Context Actions Alt+Enter
- Copy Reference Ctrl+Alt+Shift+C
- Paste Ctrl+V
- Paste from History... Ctrl+Shift+V
- Paste without Formatting Ctrl+Alt+Shift+V
- Column Selection Mode Alt+Shift+Insert
- Find Usages Alt+F7
- Refactor ▶
- Folding ▶
- Go To ▶
- Generate... Alt+Insert
- Run 'hello' Ctrl+Shift+F10
- Debug 'hello'
- Edit 'hello'...
- Show in Explorer
- File Path Ctrl+Alt+F12
- Open in Terminal
- Local History ▶
- Run File in Python Console
- Compare with Clipboard
- Create Gist...

```
hello x
PyDev console: starting.
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC
>>> runfile('C:/Users/Anonymous/PycharmProjects/GfG/hello.py
Hello World
>>>
```





Fundamentals of Python





KEYWORDS





Keywords -Definition

- Python keywords are special reserved words that have specific meanings and purposes and can't be used for anything but those specific purposes.





Number of Keywords in Python???

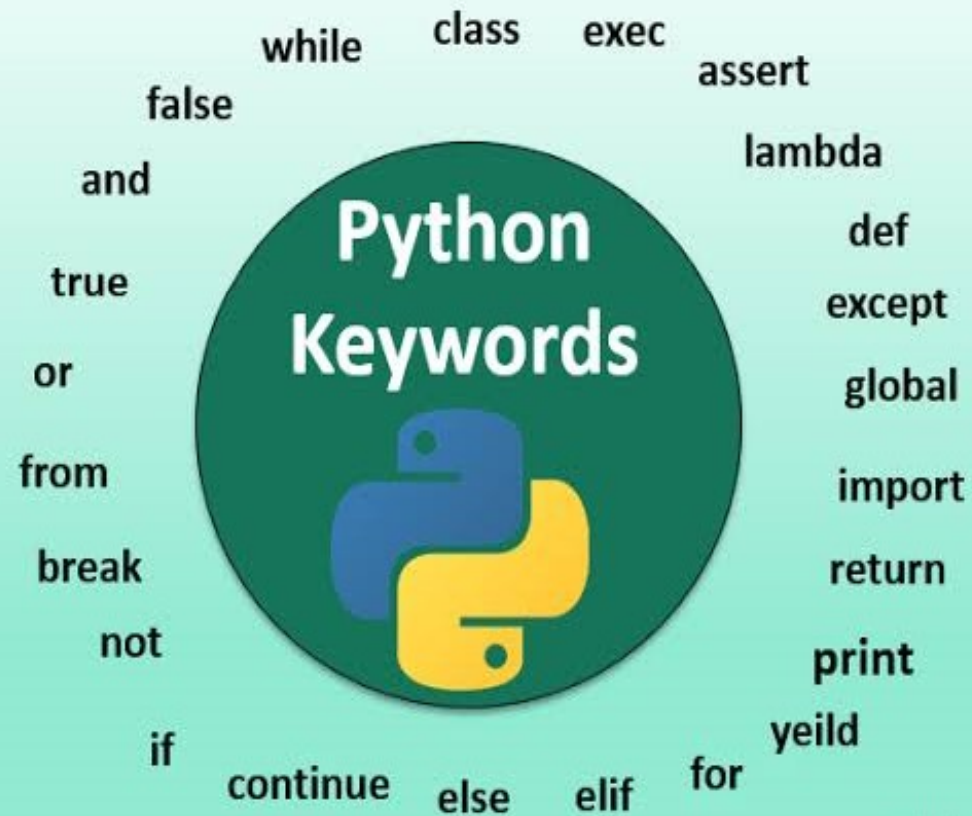
33



What are
Keywords?



Keywords



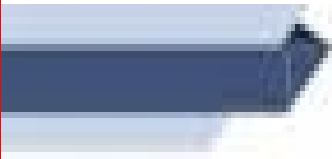


<code>and</code>	A logical operator
<code>as</code>	To create an alias
<code>assert</code>	For debugging
<code>break</code>	To break out of a loop
<code>class</code>	To define a class
<code>continue</code>	To go to the next iteration of a loop
<code>def</code>	To define a function
<code>del</code>	To delete an object
<code>elif</code>	A conditional statements, like else if
<code>else</code>	A conditional statements
<code>except</code>	Used with exceptions, what to do when an exception occurs
<code>False</code>	Boolean value
<code>finally</code>	Used with exceptions, will be executed no matter if there is an exception or not
<code>for</code>	To create a for loop
<code>from</code>	To import specific parts of a module
<code>global</code>	To declare a global variable
<code>if</code>	To make a conditional statement
<code>import</code>	To import a module
<code>in</code>	To check if a value is in a list, tuple
<code>is</code>	To test if two variables are equal
<code>lambda</code>	To create an anonymous function
<code>None</code>	Represents a null value
<code>nonlocal</code>	To declare a non-local variable
<code>not</code>	A logical operator
<code>or</code>	A logical operator
<code>pass</code>	A statement that will do nothing (null)
<code>raise</code>	To raise an exception
<code>return</code>	To exit a function and return a value
<code>True</code>	Boolean value
<code>try</code>	To make a try...except statement
<code>while</code>	To create a while loop
<code>with</code>	Used to simplify exception handling
<code>yield</code>	To end a function, returns a generator



What's
NEXT?





Let's do an
Activity!



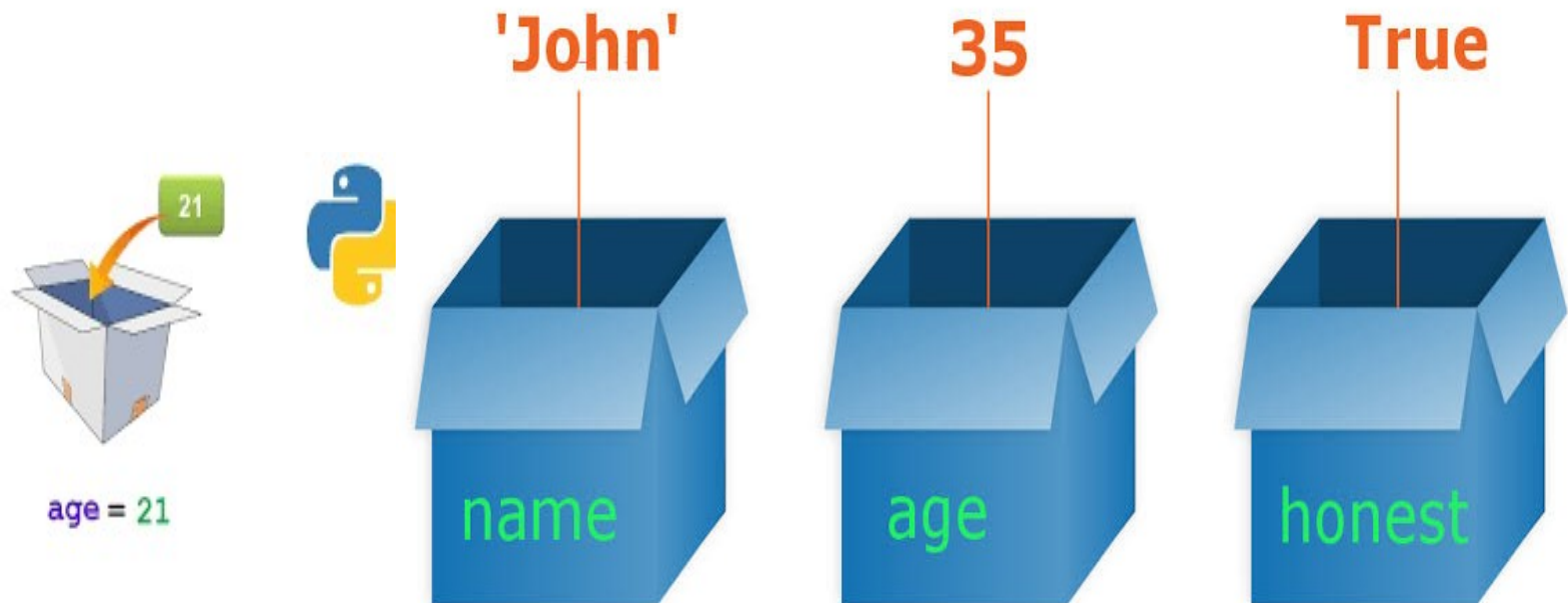


Variables



Variables

Variables are containers for storing data values.





Naming rules of Variables

- A variable can have a short name (like x and y) or a more descriptive name
- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)

Python Variables





Example

Valid/Invalid Variable Names

- | | |
|-----------------|------------------|
| ✓ 1. X | ✗ 8. int |
| ✓ 2. Name | ✗ 9. double |
| ✗ 3. first@name | ✓ 10. NAME |
| ✓ 4. name2 | ✓ 11. first_name |
| ✓ 5. _name | ✗ 12. first-name |
| ✗ 6. 2name | ✗ 13. first name |
| ✓ 7. Name | ✗ 14. v.a.l.u.e |



Multi Words Variable Names

Camel Case

Each word, except the first, starts with a capital letter:

```
myVariableName = "John"
```

Pascal Case

Each word starts with a capital letter:

```
MyVariableName = "John"
```

SnakeCase

Each word is separated by an underscore character:

```
my_variable_name = "John"
```

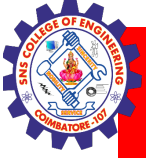


Explained

camelCase

snake_case

PascalCase



Many Values to Multiple Variables

- Python allows you to assign values to multiple variables in one line:

Example:

```
x, y, z = "Orange", "Banana", "Cherry"
```

```
Orange
```

```
Banana
```

```
Cherry
```



One Value to Multiple Variables

- And you can assign the same value to multiple variables in one line:

Example

```
x = y = z = "Orange"
```

```
print(x)
```

```
print(y)
```

```
print(z)
```

```
Orange
```

```
Orange
```

```
Orange
```



QUIZ

1. What is a valid variable name in Python?

- a) 2variable_name
- b) variable-name
- c) variableName
- d) variable name

ANSWER: C

2. Which symbol is commonly used to separate words in a variable name?

- a) -
- b) _
- c) .
- d) #

ANSWER: B



3.What should a variable name start with in Python?

- a) A number
- b) A special character
- c) A letter or an underscore
- d) A space

ANSWER:C

True or False:

Variable names in Python can include spaces.

ANSWER: FALSE

totalAmount and total_amount are equivalent in Python

ANSWER:FALSE



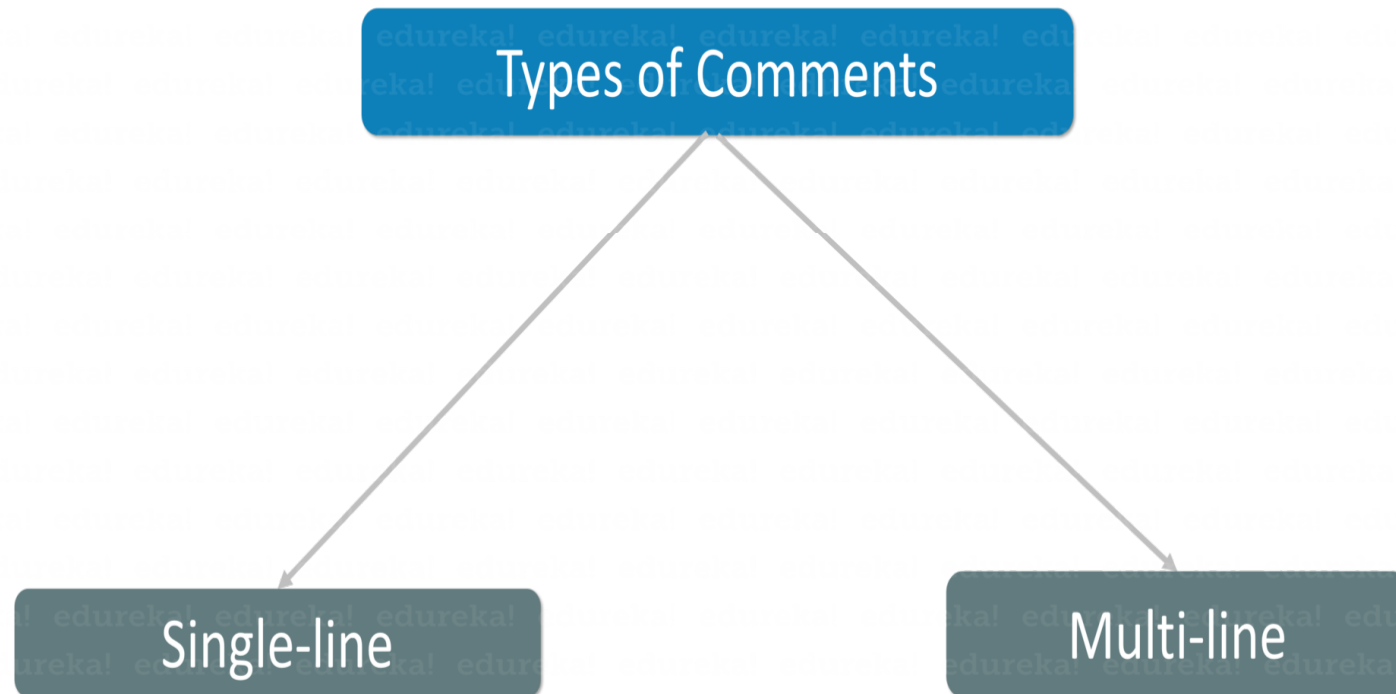
Comments

- Comments can be used to explain Python code.
- Comments can be used to make the code more readable.
- Comments can be used to prevent execution when testing code.





Types of comments





How to Use Comments in Python



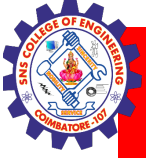


Creating a Comment

- Comments starts with a #, and Python will ignore them:

Example

```
# The value 5 is assigned to variable a  
a=5
```



Multi Line Comments

- Python does not really have a syntax for multi line comments.
- To add a multiline comment you could insert a # for each line:
- Example
- #This is a comment
- #written in
- #more than just one line
- `print("Hello, World!")`





Multiline string

- you can use a multiline string.
- Since Python will ignore string literals that are not assigned to a variable, you can add a multiline string (triple quotes) in your code, and place your comment inside it:

Example

```
"""
```

```
This is a comment
```

```
written in
```

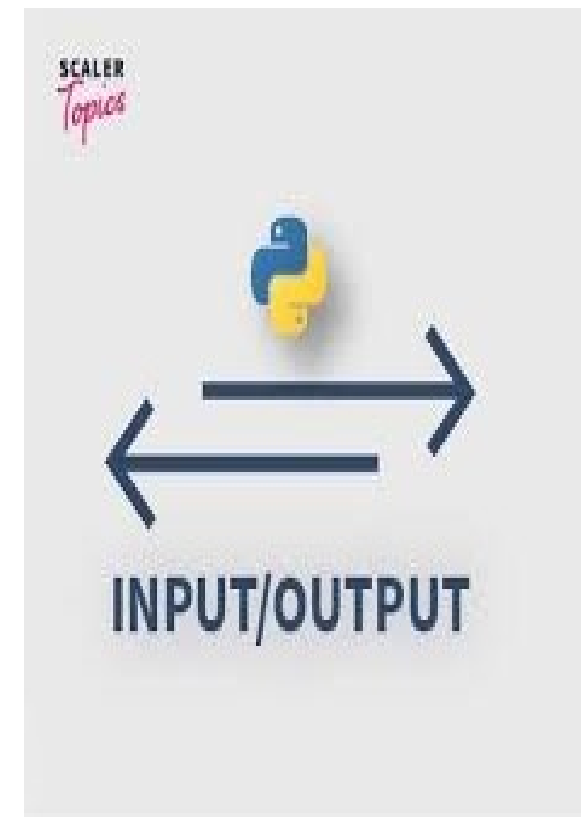
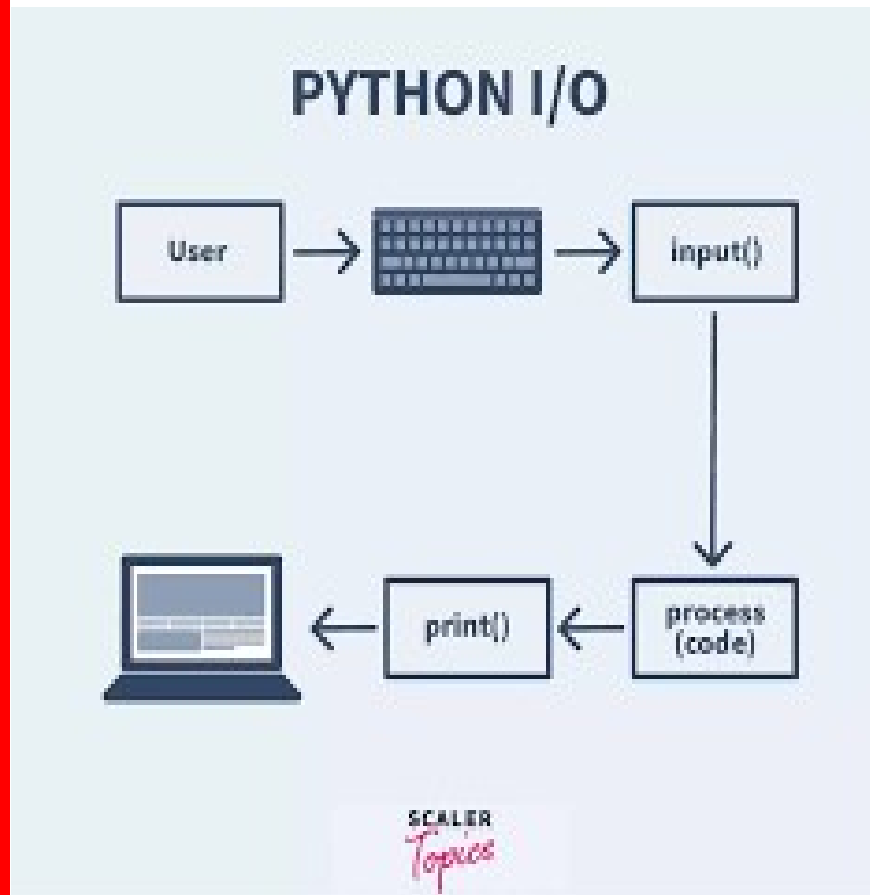
```
more than just one line
```

```
"""
```

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



Input/Output Statements





INPUT



Syntax:

```
input('prompt')
```

Example

```
# Taking input from the user
```

```
name = input("Enter your name: ")
```

```
# Output
```

```
print("Hello, " + name)
```

```
print(type(name))
```





Integer input in Python

Taking input from the user as integer

```
num = int(input("Enter a number: "))
```

```
add = num + 1
```

Output

```
print(add)
```



Output statement in Python

In Python, we can simply use the `print()` function to print output. For example,

```
print('Python is powerful')
```

```
# Output: Python is powerful
```



Escape Characters

- An escape character is a backslash \ followed by the character you want to insert.
- You will get an error if you use double quotes inside a string that is surrounded by double quotes:
- `txt = "We are the so-called \"Vikings\" from the north."`





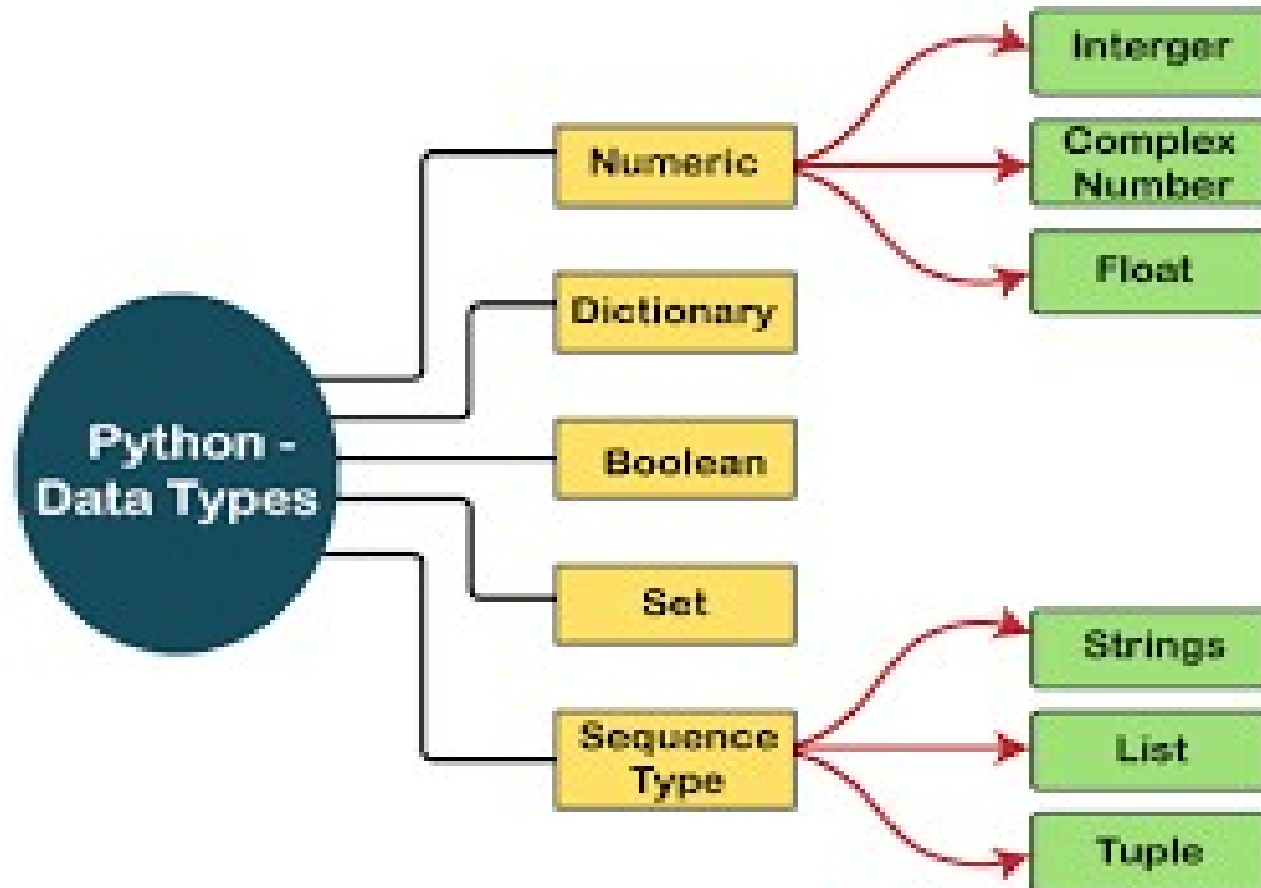
Python Escape Characters

Code	Result/Output	Description
\'	Single Quote	Add single quote with in a String
\\	Backslash	Insert single Back Slash
\n	New Line	\n takes the cursor to first position of a new line
\r	Carriage Return	\r takes the cursor to the first position of the same line
\t	Tab	\t add spaces of 8 characters
\b	Backspace	\b takes the cursor one position backward
\f	Form Feed	Form Feed is page breaking ASCII control character
\ooo	Octal value	Octal value
\xhh	Hex value	Hex value



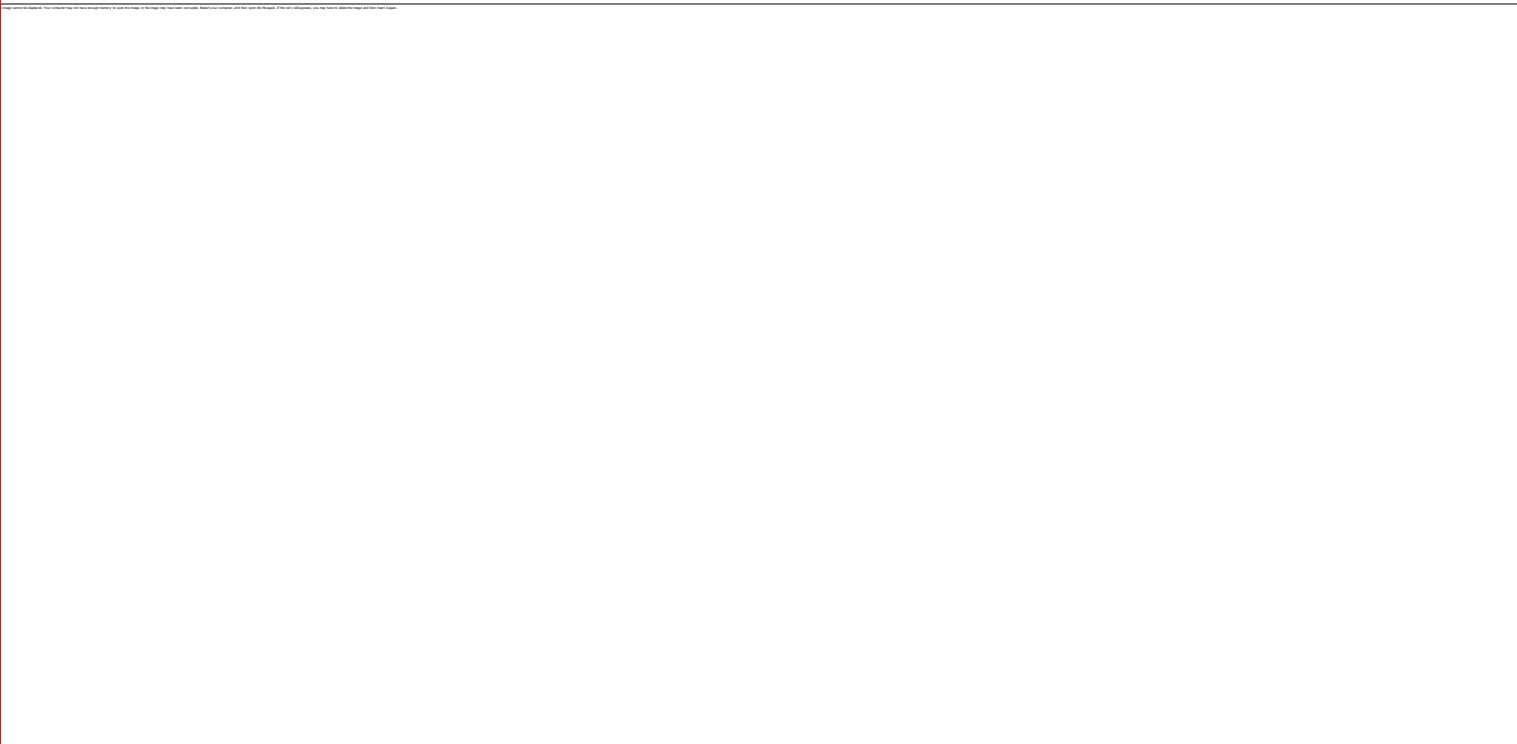


Datatypes in python

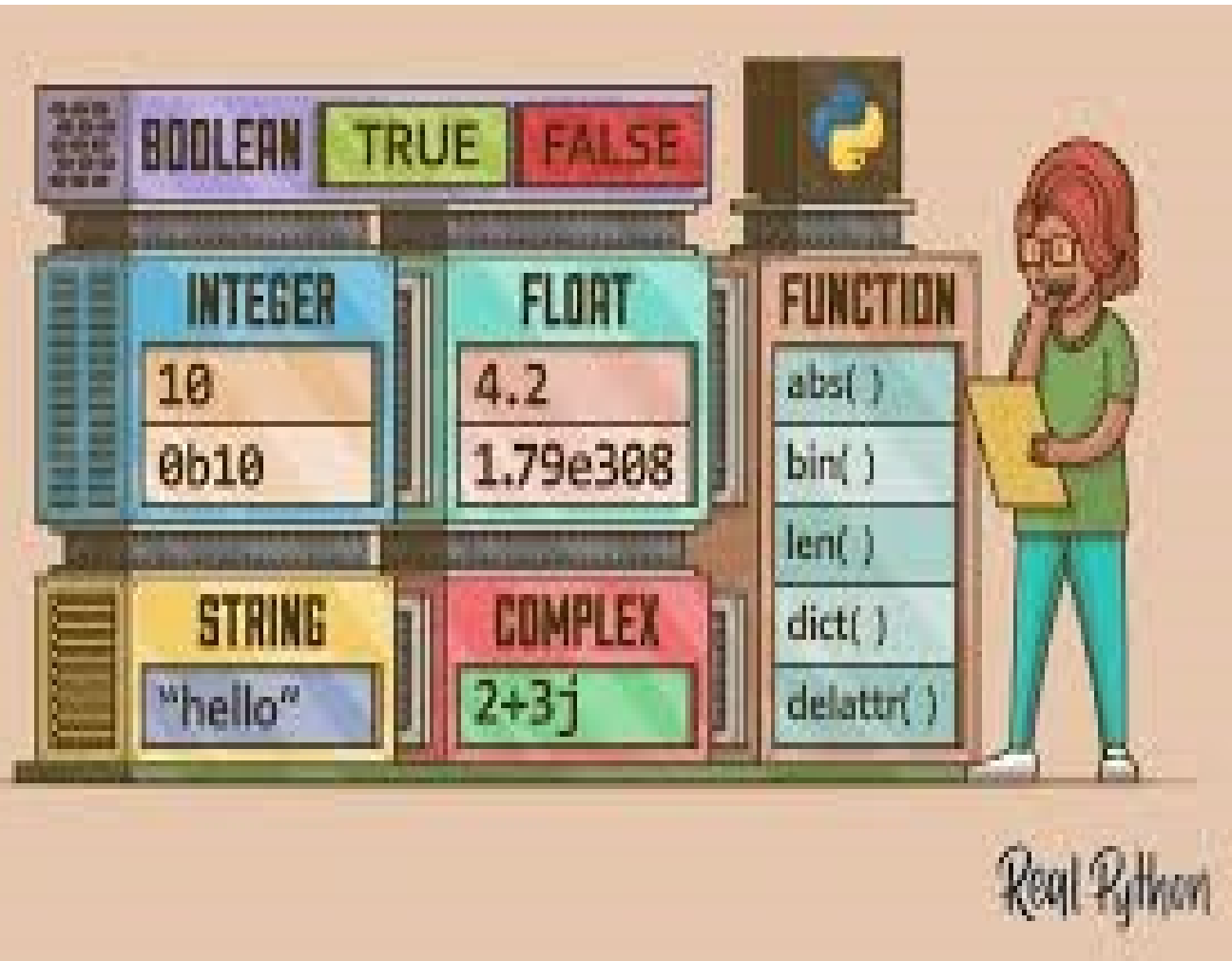




DATATYPES



Example





Getting the Data Type

- `x = 5`
`print(type(x))`



MATCH

- int → Stores textual data.
- float → A collection of unique items.
- str → Represents decimal numbers.
- bool → Represents whole numbers.
- list → Represents a true or false value.
- tuple → Immutable ordered collection of items.
- dict → An ordered collection of items.
- set → A collection of key-value pairs.



ANSWER

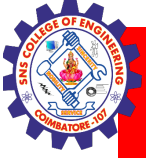
- int → Represents whole numbers.
- float → Represents decimal numbers.
- str → Stores textual data.
- bool → Represents a true or false value.
- list → An ordered collection of items.
- tuple → Immutable ordered collection of items.
- dict → A collection of key-value pairs.
- set → A collection of unique items.



Setting the Data Type

Example	Data Type	Try it
<code>x = "Hello World"</code>	str	Try it »
<code>x = 20</code>	int	Try it »
<code>x = 20.5</code>	float	Try it »
<code>x = 1j</code>	complex	Try it »
<code>x = ["apple", "banana", "cherry"]</code>	list	Try it »
<code>x = ("apple", "banana", "cherry")</code>	tuple	Try it »
<code>x = range(6)</code>	range	Try it »
<code>x = {"name" : "John", "age" : 36}</code>	dict	Try it »
<code>x = {"apple", "banana", "cherry"}</code>	set	Try it »
<code>x = frozenset({"apple", "banana", "cherry"})</code>	frozenset	Try it »
<code>x = True</code>	bool	





Boolean Values

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

OUTPUT

```
True  
False  
False
```



TYPE Casting

INTEGERS

```
x = int(1) # x will be 1  
y = int(2.8) # y will be 2  
z = int("3") # z will be 3
```

FLOAT

```
x = float(1) # x will be 1.0  
y = float(2.8) # y will be 2.8  
z = float("3") # z will be 3.0  
w = float("4.2") # w will be 4.2
```

STRINGS

```
x = str("s1") # x will be 's1'  
y = str(2) # y will be '2'  
z = str(3.0) # z will be '3.0'
```

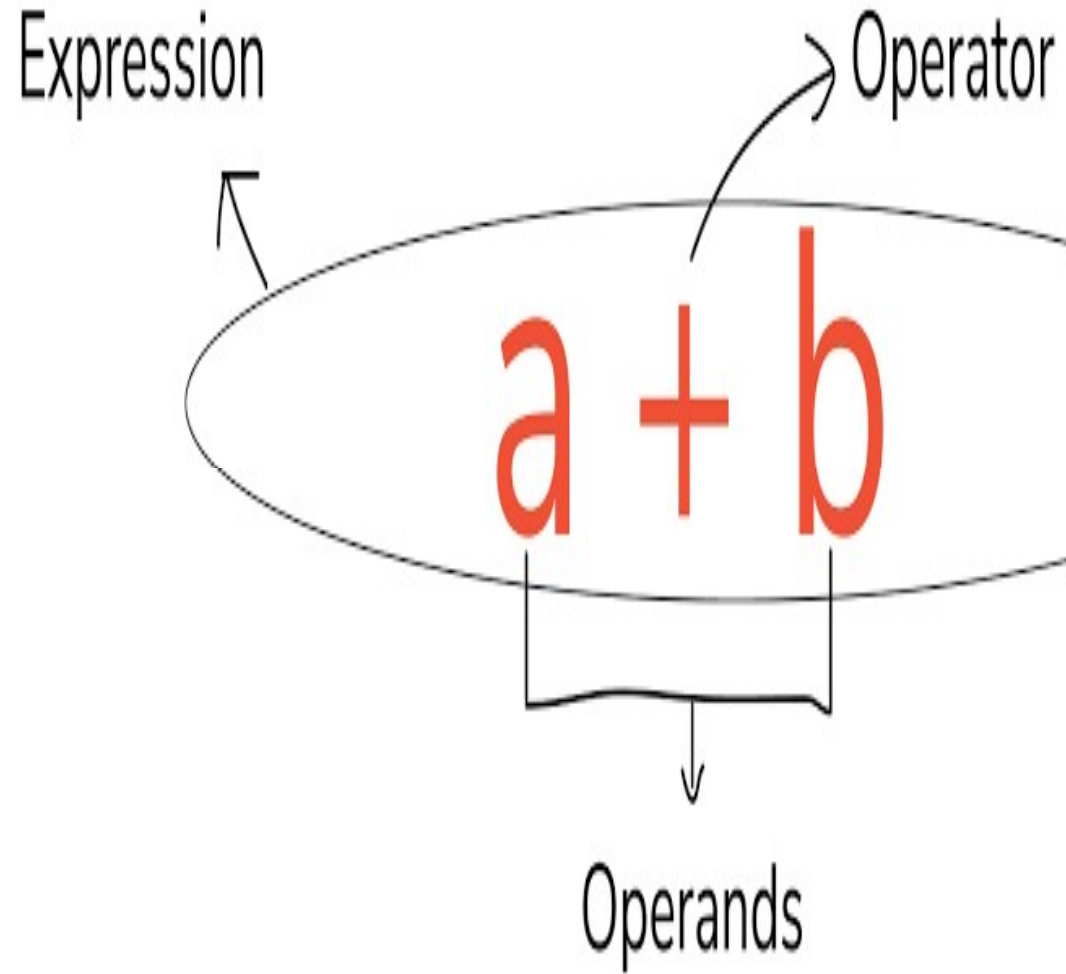


Python operators

 Desant Technologies

Python Operators





OPERATORS

- An operator is a symbol that represents an operations that may be performed on one or more operands.
- An operand is a value that a given operator is applied to.
- **Example: $4+(3*k)$**
+, * are operator and 4,3,k are operands



Operators and Operands

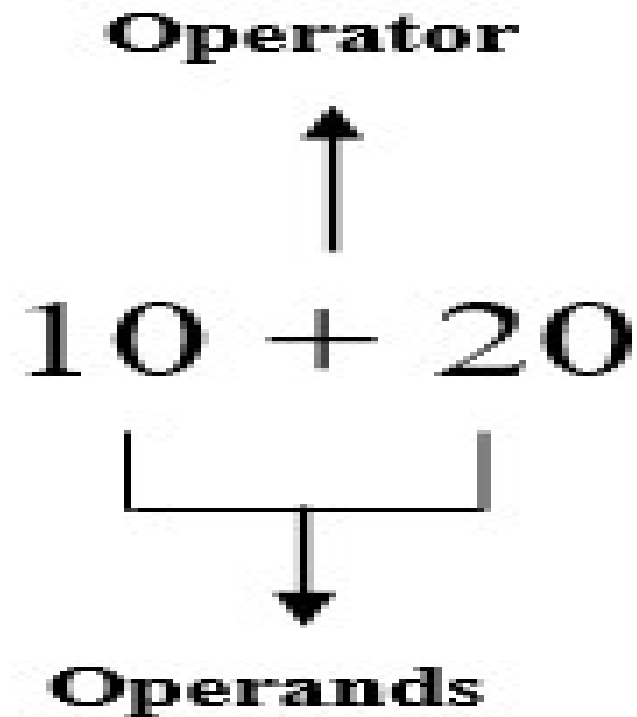


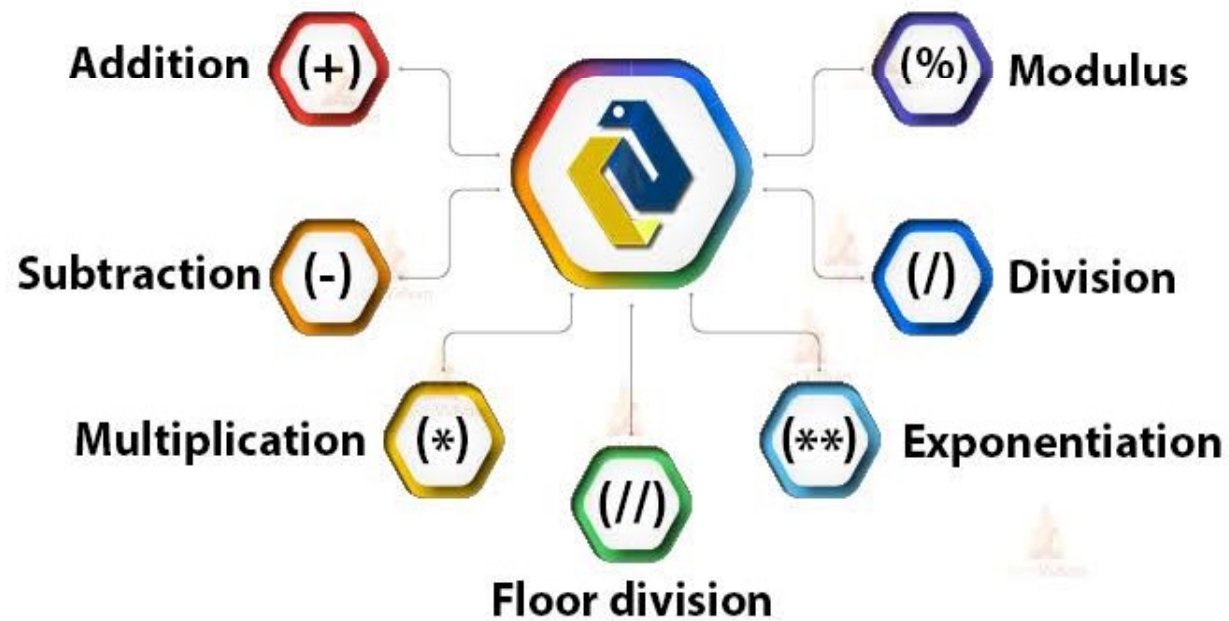
Fig: Operator and operands

Types of Operators



Arithmetic Operators

Python Arithmetic Operators



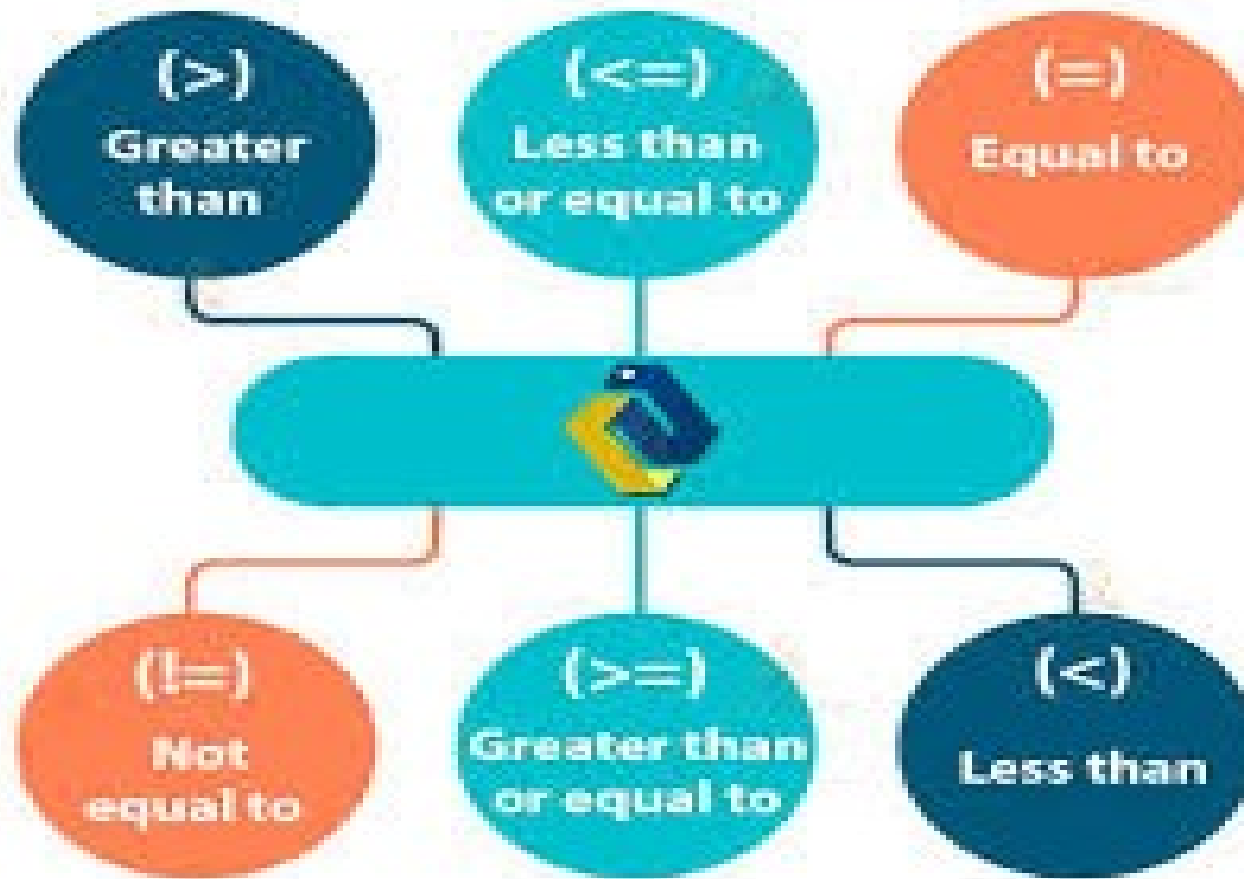


Arithmetic operators

Operator	Name	Description	Syntax	Example
+	Addition	Performs addition	$c = a + b$	$a = 5, b = 5$ then $c = 10$
-	Subtraction	Performs subtraction	$c = a - b$	$a = 5, b = 3$ then $c = 2$
*	Multiplication	Performs multiplication	$c = a * b$	$a = 5, b = 5$ then $c = 25$
/	Division	Performs division	$c = a / b$	$a = 10, b = 5$ then $c = 2$
%	Modulus	Performs division but returns the remainder	$c = a \% b$	$a = 15, b = 2$ then $c = 1$
//	Floor Division	Performs division but returns the quotient in which the digits after the decimal points are removed	$c = a // b$	$a = 15, b = 2$ then $c = 7$
**	Exponent	Performs multiplication to power raised	$c = a ** b$	$a = 2, b = 4$ then $c = 16$

Relational Operators

PYTHON RELATIONAL OPERATORS





Example

Operator	Name	Description	Syntax	Example
>	Greater than	Compares the operands and then returns True if the left operand is greater than the right or else False .	$a > b$	$a = 15, b = 5$ then True
<	Lesser than	Compares the operands and then returns True if the left operand is lesser than the right or else False	$a < b$	$a = 5, b = 15$ then True
==	Equal to	Compares the operands and then returns True if both the operands are equal or else False	$a == b$	$a = 5, b = 5$ then True
!=	Not equal to	Compares the operands and then returns True if both the operands are not equal or else False	$a != b$	$a = 10, b = 5$ then True
>=	Greater than or equal to	Compares the operands and then returns True if the left operand is greater than or equal to the right or else False	$a >= b$	$a = 15, b = 2$ then True
<=	Lesser than or equal to	Compares the operands and then returns True if the left operand is lesser than or equal to the right or else False	$a <= b$	$a = 2, b = 15$ then True



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(//=)



Example

Operator	Example	Equivalent To
Assignment =	$i = 1$	$i = 1$
Addition Assignment +=	$j += 1$	$j = j + 1$
Subtraction Assignment -=	$k -= 2$	$k = k - 2$
Multiplication Assignment *=	$m *= 2$	$m = m * 2$
Float Division Assignment /=	$n /= 2$	$n = n / 2$
Integer Division Assignment //=	$p //= 2$	$p = p // 2$
Modulus or Remainder Assignment %=	$q %= 2$	$q = q \% 2$
Exponent Assignment **=	$r **= 2$	$r = r ** 2$



Logical Operators

PYTHON LOGICAL OPERATORS

1	and
2	or
3	not

Logical Operators in Python

There are three logical operators in Python

AND OPERATOR

Returns True if both of the operands are True; False otherwise.

Example: `(4>2) and (3>6)` returns False

OR OPERATOR

Returns True if either one or both of the operands is True; False otherwise.

Example: `(4>2) and (3>6)` returns True

NOT OPERATOR

Returns True if the given expression or operand is False and vice-versa.

Example: `not(3>6)` will return True

Logical Operators

Python - Logical Operators

- not

x	not x
False	True
True	False

- and

x	y	x and y
False	False	False
False	True	False
True	False	False
True	True	True

- or

x	y	x or y
False	False	False
False	True	True
True	False	True
True	True	True



Operator Priority

<http://inderpsingh.blogspot.com/>

Example

Example of Logical Operator

```
print("Logical Operator")  
print(10<5 and 10<20)  
print(10<5 or 10<20)  
print(not(10<20))
```

Output:

Logical Operator
False
True
False

Example

Operator	Meaning	Example	Result
and	Logical and	$(5 < 2)$ and $(5 > 3)$	False
or	Logical or	$(5 < 2)$ or $(5 > 3)$	True
not	Logical not	not $(5 < 2)$	True



Membership Operators

PYTHON MEMBERSHIP OPERATORS





Membership operators

MEMBERSHIP OPERATORS IN PYTHON

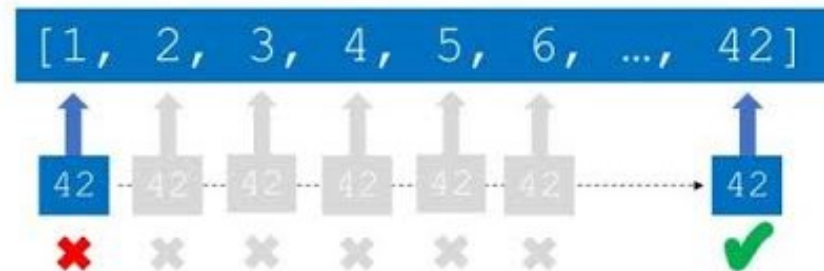
in	Return True if the value or variable exist in the given sequence; False otherwise
not in	Return True if the value or variable does not exist in the given sequence; False otherwise

Example

`finxter`

Python Membership “in” operator

```
>>> item = 42  
>>> my_list = [1, 2, 3, 4, 5, 6, ..., 42]  
>>> item in my_list  
True
```





Identity operators

PYTHON IDENTITY OPERATORS





Identity operators

Identity Operators in Python

is operator

Return True if both the operands or variables are referring to the same memory location. Otherwise, it will return False

Example: a is b

is not operator

Return True if both the operands or variables are referring to the different memory location. Otherwise, it will return False.

Example: a is not b

Example

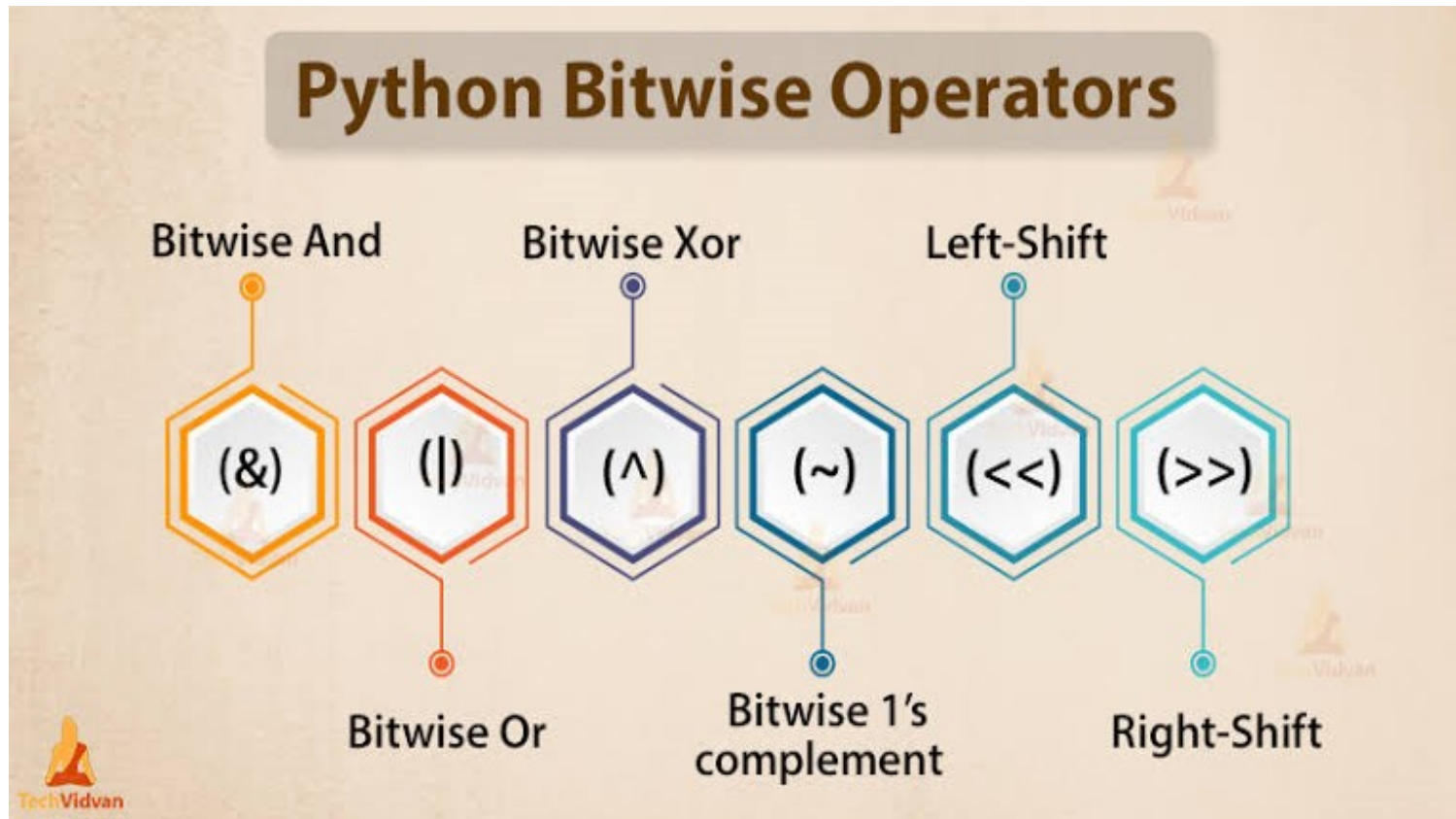
```
A = 20  
B = 20  
C = 22  
D = 15  
E = [10, 22]  
F = [10, 22] #E and F are equal but are not same object
```

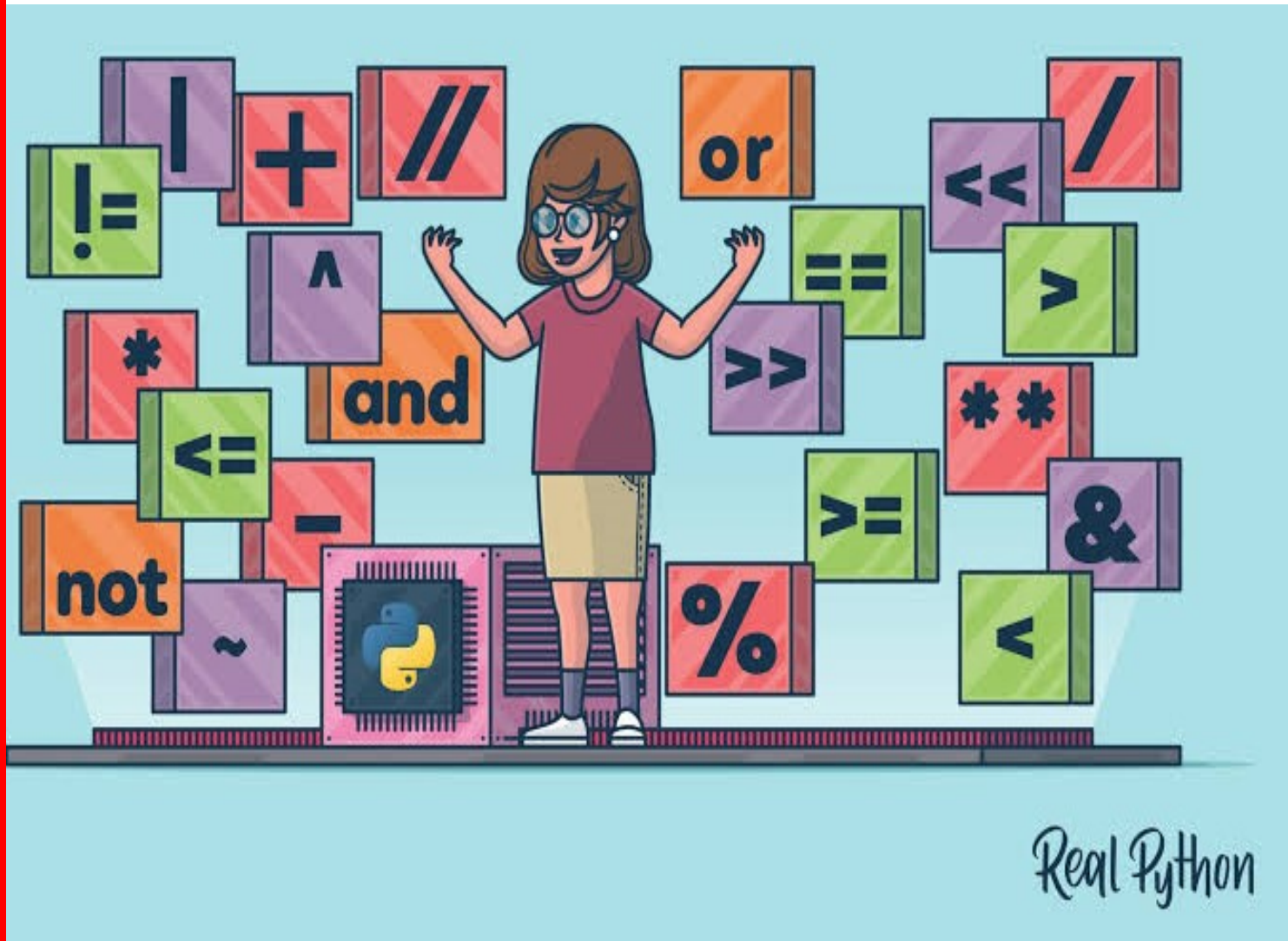
```
print(A is B)  
print(C is B)  
print(E is not F)
```

```
>>> runfile('C:/Users/Aditya/PycharmProjects/Hello1/Hello1.py')
```

```
True  
False  
True
```

Bitwise operators





Real Python



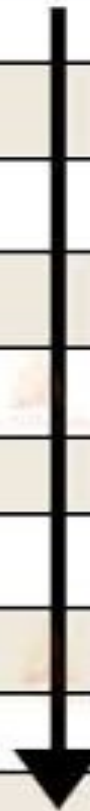


Operator precedence



Python Operator Precedence

Precedence	Operator Sign	Operator Name
Highest	**	Exponentiation
	+X, -X, ~X	Unary positive, unary negative, bitwise negation
	*, /, //, %	Multiplication, division, floor, division, modulus
	+, -	Addition, subtraction
	<<, >>	Left-shift, right-shift
	&	Bitwise AND
	^	Bitwise XOR
		Bitwise OR
	==, !=, <, <=, >, >=, is, is not	Comparison, identity
	not	Boolean NOT
	and	Boolean AND
Lowest	or	Boolean OR





Bitwise operators

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



WORD SEARCH

ASSIGNMENTROADI
CARRYEOIORRDAEO
COMPARISONLOGIM
ARIIALAARBITORS
RLNADM SERIELSTN
LTETERTIOSLGATO
AEEOSMTMITOTOST
OILIONIAOLLALHE
GNNIMNITADARSSR
EPAESSTRATTALAR
OSSOCITARSCHRAT
NRABOSCAECADINT

ASSIGNMENT
CARRY
COMPARISON
LOGICAL
ARITHMETIC
RELATIONAL
BITWISE
UNARY
TERNARY
INCREMENT
DECREMENT



· ASSIGNMENT (row 1, starting at column 1 to 10)

· CARRY (row 2, starting at column 3 to 7)

· COMPARISON (row 3, starting at column 2 to 11)

· LOGICAL (row 3, starting at column 8 to 15)

· ARITHMETIC (row 4, starting at column 1 to 11)

· RELATIONAL (row 5, starting at column 5 to 14)

· BITWISE (row 4, starting at column 8 to 15)

· UNARY (row 7, starting at column 2 to 6)

· TERNARY (row 10, starting at column 3 to 9)

· INCREMENT (row 9, starting at column 5 to 14)

· DECREMENT (row 9, starting at column 8 to 17)



```
main.py  [Refresh] [Settings] Save Run Output Clear
3 b = 5
4 x = True
5 y = False
6 c = 3
7 d = 10 # 1010 in binary
8 e = 4 # 0100 in binary
9 f = [1, 2, 3]
10 h = 2
11 print("Arithmetic Operator:")
12 print(f"{a} + {b} = {a + b}") #
13 print("\nComparison Operator:")
14 print(f"{a} > {b} -> {a > b}")
15 print("\nLogical Operator:")
16 print(f"x or y -> {x or y}")
17 print("\nAssignment Operator:")
18 c += 2
19 print(f"c += 2 -> {c}")
20 print("\nBitwise Operator:")
21 print(f"{d} & {e} -> {d & e}")
22 print("\nIdentity Operator:")
23 print(f"f is [1, 2, 3] -> {f is [1, 2, 3]}") #
24 print("\nMembership Operator:")

Arithmetic Operator:
10 + 5 = 15

Comparison Operator:
10 > 5 -> True

Logical Operator:
x or y -> True

Assignment Operator:
c += 2 -> 5

Bitwise Operator:
10 & 4 -> 0

Identity Operator:
f is [1, 2, 3] -> False

Membership Operator:
2 in [1, 2, 3] -> True

=== Code Execution Successful ===
```





Thank You