Python Programming

Python Fundamentals

Identifier

- valid word which is used to perform an action
- most of the times the identifiers are lower cased
- can be
 - o variable name
 - function name
 - constant
 - o class name
 - keyword
- rules
 - o can not start with number
 - e.g.
 - 1name is invalid identifier
 - one_name is valid identifier
 - o can not contain special character like space
 - e.g.
 - first name is invalid identifier
 - first_name is valid identifier
 - o may use only underscore (_)
- conventions
 - o for variables: lower case
 - e.g. name, address, first_name
 - o for constant: upper case
 - e.g. PI, INTEREST_RATE
 - o for functions: lower case with underscore



- e.g. is_eligible_for_voting
- for class: lower case with first letter uppercase
 - e.g. Person, Mobile

Variable

- identifier used to store a value
- variable can not be declared explicitly
- syntax

<variable name> = <initial value>

• e.g.

num = 100

- Python doesn't have any feature for constant values.
- Python relies on convention for the same. The constant variables are declared all caps.

PI = 3.1415

Data types

- Data type is an attribute/information associated with a piece of data that tells a computer system how to interpret its value
- Understanding data types ensures that data is collected in the preferred format and the value of each property is as expected
- It helps to know what kind of operations are often performed on a variable
- In python, all data types are inferred
 - Data types are assigned implicitly
 - Data types will get assigned automatically (by Python itself) by looking at the CURRENT value of the variable
- Can not declare a variable with explicit data type

```
# can not declare explicit
# int num = 100 # error
```

- Literal (a.k.a. constant) is a raw data given in a variable or constant
- Python Data Types
 - o int
 - represents the whole numbers (+ve or -ve)
 - e.g.
 - num = 100
 - myvar = -10
 - Literals: 65, 0x41, 0o101, 0b100001
 - float
 - represents a value with decimal
 - e.g.
 - salary = 1034.60
 - Literals: 3.14, -9.81
 - o str
 - represents a string
 - to create a string value use
 - single quotes
 - used to create single line string
 - e.g.

name = 'steve

- double quotes
 - used to create single line string
 - e.g.

```
last_name = "jobs"
```

- tripe double quotes
 - used to create multi-line string
 - e.g.

```
address = """

House no 100,

XYZ,

pune 411056,

MH, India.
"""
```

o bool

- represents boolean value
- can contain one of the two values [True/False]
- e.g.

```
can_vote = True
```

■ Literals: True, False

• complex

 \blacksquare num = 2 + 3j

object

- stores any type.
- Literal: None (represents Nothing)

collection types

mylist = [11, 22, 33, 44]

- mytuple = ('A', 'B', 'C')
- myset = { 2.2, 4.4, 6.6 }
- mydictionary = { 'A': 'Apple', 'B': 'Ball' }

Type Hinting

- Python is dynamic typing i.e. types are assigned dynamically.
- Hard for developers to guess type of variable in code
- Type hints increase readability of the program.

```
num: int = 123
```

- Helpful for programmers, IDEs, and type checkers to identify potential errors
- Also, helps IDE to provide intellisense.
- NOTE: Providing hint doesn't change the type. Types are still inferred by the Python at runtime.

Type conversion

- Implicit type conversion
 - Automatically done by compiler if possible (avoids data loss)
 - \circ res1 = 5 + 3.14
 - 3.14 = float
 - 5 = int -> float
 - res1 = float
 - \circ res2 = True + 20

- True = bool --> int
- 20 = int
- \blacksquare res2 = int

• Explicit type conversion

- Programmer converts explicitly using functions like int(), float(), str(), etc.
- Examples:

```
f = 9.81; i = int(f)
```

```
s = "3.14"; f = float(s)
```

```
i = 103; b = bool(i)
```

```
x = 108; s = str(x)
```

```
s = "False"; b = bool(s) # b is True
```

- If conversion is not compatible, runtime error will raise.
 - o Examples:

```
c = 2 + 3j
n = int(c)  # error
```

```
s = "One"
n = int(s) # error
```

```
n = 108
s = "string" + n # error: implicit conversion not done
s = "string" + str(n) # okay
```

• Conversion functions for collections as follows: list(), tuple(), set(), dict()

User Input

- input() function returns string entered by end user.
- Programmer can use explicit type casting to convert string to other type.

```
name = input("Enter Name: ")
roll = int(input("Enter Roll: "))
marks = float(input("Enter Marks: "))
```

Operators

Arithmetic

- + : addition/string concatination
- -: subtraction
- *: multiplication
- %: modulus
 - % -- gives remainder e.g. 22 % 5 = 2
- /: true division (float)

- / -- true division e.g. 22 / 5 = 4.4
- //: floor division (int)
 - // -- floor division e.g. 22 / 5 = 4 (value after decimal is truncated)
- **: power of
 - ** -- power e.g. 7 ** 3 = 7 * 7 * 7 = 343

Assignment

- = : assignment
 - o var = value -- right side value assigned to left side var
- +=, -=, *=, /= : short-hand assignment
 - o var += value --> var = var + value (short-hand or inplace add)
 - o var += 1 --> var = var + 1 (No ++ and -- operators like C/Java)

Comparison

- == : equal to
- != : not equal
- > : greater than
- < : less than
- >=: greater than or equal to
- <=: less than or equal to
- Used for condition: Evaluated as bool (True/False)
 - o 22 > 7 --> True
 - o 22 < 7 --> False

Logical

- and:
 - logical and operator
 - o returns true only when both the conditions are true

- o rule
 - true and true => true
 - true and false => false
 - false and true => false
 - false and false => false

```
if (age > 20) and (age < 60):
    print(f"{age} is within the limit")
else:
    print(f"{age} is not within the limit")</pre>
```

- or:
 - o logical or operator
 - o returns true when one of the conditions is true
 - o rule
 - true or true => true
 - true or false => true
 - false or true => true
 - false or false => false

```
if (age < 18) or (age > 70):
    print(f"{age} is too young or too old")
else:
    print(f"{age} is not yound or old")
```

- not:
 - logical not operator
 - returns complement of the condition

- o rule
 - not true => false
 - not false => true

Bitwise operators

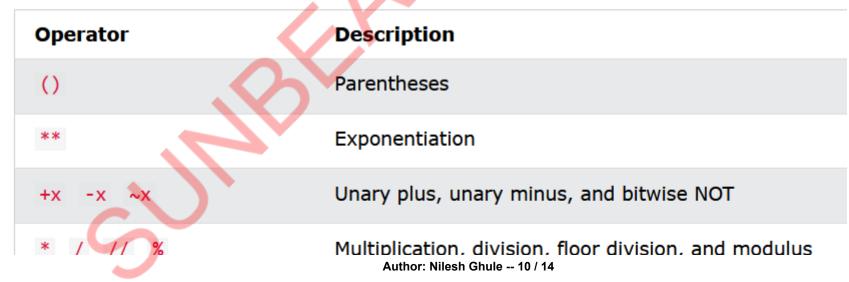
- ~ bitwise not
- & bitwise and
- | bitwise or
- ^ bitwise xor
- << bitwise left shift
- >> bitwise right shift

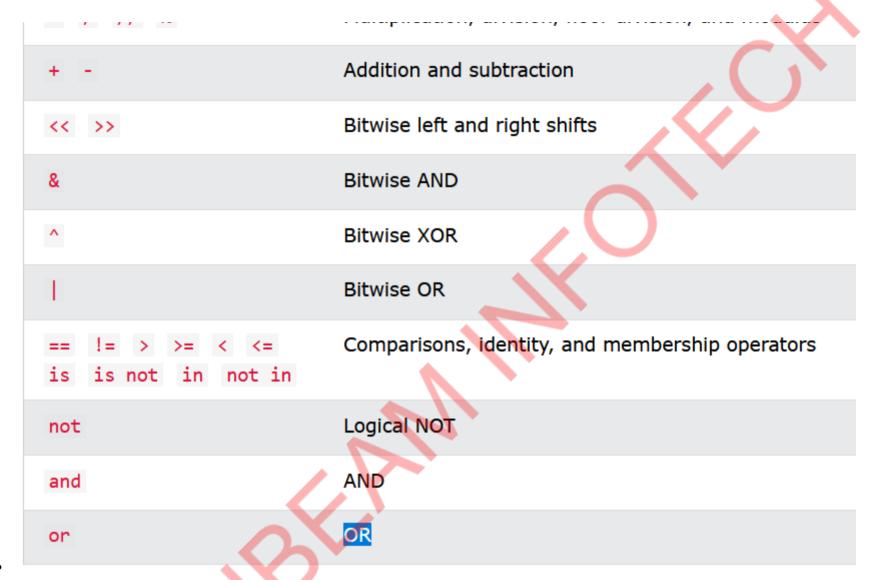
Special operators

- is, is not identity operators
- in, not in membership operators

Precedence

• When multiple operators used in same expression, they are solved by their Precedence.





Keywords

- reserved identifiers by Python
- can not use keyword for declaring variables or functions
- e.g. if, elif, else, for, while, switch

- pass
 - o do not do anything
 - o pass the control to the next line
 - used to create empty function/class
- def
 - o used to define a function
- return
 - used to return a value

Python control statements

- Python has multiple control statements
 - o if-else statement
 - while loop
 - for loop
 - match statement
- All control statements have block of statements.
- Python indentation
 - o Indentation defines Block of statements
 - Indention is not merely for readability of the code.
 - Must have same number of spaces for the block.
 - If number of spaces mismatch/mistaken, Python compiler raise error.

if-else statement

- Decision control statement based on the condition
- Various syntaxes:

single if
if condition:
 block

Author: Nilesh Ghule -- 12 / 14

```
# if-else
if condition:
   block
else:
   block
# nested if-else
if condition:
   block
else:
   if condition:
       block
   else:
       block
# if-elif-else
if condition:
   block
elif condition:
   block
else:
   block
# empty if
if condition:
   pass
```

```
# short-hand if
if condition: statement
```

```
# short-hand if-else
statement1 if condition else statement2
    # statement1 executed if condition is true
    # otherwise statement2 is executed
```

- In Python, conditions are evaluated to True/False. Typically conditions are given using comparison and logical operators.
- Note that, int/string values are implicitly converted to bool when used in condition context.