python-notes

April 14, 2024

0.1 Author: Umar Hayat Khan Lodhi

0.2 Subject: Python Notes

0.3 Contact: lodhiumar2023@gmail.com

0.4 Github: https://github.com/lodhiumar2023

1 Variable in Python

Data/ Values can be stored in temporary storage spaces called variables

[56]:
```
# os is a variable

os =  "Windows"
print(os)
```

```
Windows
```

2 Each variable is associated with a data-type

1. int (Numbers values)
2. float (Numbers with decimal values)
3. Bolean (True, Flase, 1, 0)
4. string (Data wtih quores 'Windows', "Wnidows", " 'Windows' ")
5. complex (Real Part and Imaginary Party 3+4j) in python j presents i

[ ]:
```
integer_data = 10
print(integer_data)
print(type(integer_data)) # to check data type
```

```
10
<class 'int'>
```

[ ]:
```
float_data = 10.2
print(float_data)
print(type(float_data))
```

```
10.2
<class 'float'>
```

[ ]:
```
bolean_data = True
print(bolean_data)
print(type(bolean_data))
```

```
True
<class 'bool'>
```

[ ]:
```
string_data = "Windows"
print(float_data)
print(type(string_data))
```

```
10.2
<class 'str'>
```

[ ]:
```
# 3 is real part and 4j is imaginary part
complex_data = 3 + 4j
print(float_data)
print(type(complex_data))
```

```
10.2
<class 'complex'>
```

3 Operators in Python

Operators are used to perform operations on variables and values.

1. Arithmetic Operators (+, -, /, *)
2. Relational Operators (<, >, ==, !=)
3. Logical Operators (and &, or |)

4 Arithmetic Operators (+, -, /, *)

[ ]:
```
a = 20
b = 10

print(a+b) # Addition
print(a-b) # Subtraction
print(a/b) # Division
print(a*b) # Multiplication
```

```
30
10
2.0
200
```

5 Relational Operators (<, >, ==, !=)

```
[ ]: a = 20
     b = 10

     print(a<b)  # Less Than
     print(a>b)  # Greater Than
     print(a==b) # Equal to
     print(a!=b) # Not equal to
```

```
False
True
False
True
```

6 Logical Operators (and &, or |)

```
[ ]: a = True
     b = False

     # When the value of two operations is "True" so output will be "True",␣
      ↪otherwise "False".
     print(a & a)
     print(a & b)
     print(b & b)

     print()

     # When value of an operation is "True" so output will be "True".
     print(a | a)
     print(a | b)
     print(b | b)
```

```
True
False
False

True
True
False
```

7 Python Token

Smallest meaningful compnents in a program 1. **Keywords** (Keywords are special reserved words and cannot be used as variable)

2. **Identifier** (Identifiers are names used for variables, functions, or objects)

RULES

a. No special character except _(underscore)

b. Identifier are case sensitive.

c. First character cannot be a digit.

3. **Literals** (Literals are constents, means stored value in variables are literals)
4. **Operators**

a. Arithmetic Operators (+, -, /, *)

b. Relational Operators (<, >, ==, !=)

c. Logical Operators

8 Working with Strings in Python

Strings are sequence of characters enclosed within single quotes('‘), double qoutes("“), or triple quotes("' "‘)
a. 'Hellow World'
b."Hellow World"
c.‘'Hellow World"' (Used for multi lines strings)

[ ]:
```
single_quotes_string = 'This is a single quotes string'
double_quotes_string = "This is a double quotes string"
triple_quotes_string = '''
These are multi lines
These are multi lines
These are multi lines
'''

print(single_quotes_string)
print(double_quotes_string)
print(triple_quotes_string)
```

```
This is a single quotes string
This is a double quotes string

These are multi lines
These are multi lines
These are multi lines
```

9 Extract individual character using indexes or indices.

[ ]:
```
# In python index strated from 0
# for example
# A = 0
# B = 1
# C = 2
# D = 3

string = "ABCD"
print(string)
```

```
ABCD
```

[ ]:
```
# Extracting characters.

character_1_extracting = string[0]
print(character_1_extracting)

print() # for blank row

character_2_extracting = string[1]
print(character_2_extracting)

print() # for blank row

character_3_extracting = string[2]
print(character_3_extracting)

print() # for blank row

character_4_extracting = string[3]
print(character_4_extracting)
```

```
A

B

C

D
```

[ ]:
```
# To extract more than one character.

more_characters_extracting = string[0:2] # Index of character 2 is exclusive.
print(more_characters_extracting)
```

```
AB
```

[ ]:
```
# To extract character from reverse order.

reverse_character_1 = string[-1]
print(reverse_character_1)
print() # for blank row

reverse_character_2 = string[-2]
print(reverse_character_2)

print() # for blank row

reverse_character_3 = string[-3]
print(reverse_character_3)

print() # for blank row

reverse_character_4 = string[-4]
print(reverse_character_4)
```

```
D

C

B

A
```

10 String Functions

[ ]:
```
# Finding length of a string

len(string)
```

[ ]:
```
4
```

[ ]:
```
# Convert a string in lower case.

first_program = "THIS IS MY FIRST PROGRAM"
lower_case_string = first_program.lower()

print(lower_case_string)
```

```
this is my first program
```

[ ]:
```
# Convert a string in upper case.

first_program = "this is my first program"
```

```
upper_case_string = first_program.upper()

print(upper_case_string)
```

```
THIS IS MY FIRST PROGRAM
```

[ ]:
```
# Convert a string in title case.

first_program = "this is my first program"
title_case_string = first_program.title()

print(title_case_string)
```

```
This Is My First Program
```

[ ]:
```
# Swap case a string.

first_program = "THIS IS MY FIRST PROGRAM"
swap_case_string = first_program.swapcase()

print(swap_case_string)
```

```
this is my first program
```

[ ]:
```
first_program = "this is my first program"
swap_case_string = first_program.swapcase()

print(swap_case_string)
```

```
THIS IS MY FIRST PROGRAM
```

[ ]:
```
# Replace a substing. (In replace function a character or a word can be␣
 ↪changed).

about_me = "My favourite fruit is Mango"
print(about_me)

print() # to insert a blank line.

about_me.replace("Mango", "Apple")
```

```
My favourite fruit is Mango
```

[ ]:
```
'My favourite fruit is Apple'
```

[ ]:
```
# To count number of occurance of substring, can be character or a word.
```

```
student_list = ("Kashif", "Asif", "Nasir", "Yasir", "Kashif", "Asif")
count_substring = student_list.count("Kashif")
print(count_substring)

print() # to insert a blank line.

fruit_name = "Banana"
count_substring = fruit_name.count("a")
print(count_substring)
```

```
2

3
```

[ ]:

```
# To find the index of a substring.

about_me = "My favourite fruit is Mango"
print(about_me)

print() # to insert a blank line.

about_me.find("favourite")
```

```
My favourite fruit is Mango

```

[ ]: 3

[ ]:

```
# Split a string using a special character or any entity.

course_details = "Contents of this course is containing on, python, numpy,␣
 ↪pandas"
print(course_details)

course_details.split(',')
```

```
Contents of this course is containing on, python, numpy, pandas
```

[ ]:

```
['Contents of this course is containing on', ' python', ' numpy', ' pandas']
```

11 Data Types in Python

1. Tuple
2. List
3. Dictionary
4. Set

11.0.1 1. Tuple

Tuple is an order collection of elements enclosed within ( ), elements can be different types (heterogeneous)
Note: Tuples are immmutable (Stored value in Tuple cannot be changed)

[ ]:
```
tup1 = (1, "a", True, "b", False)
print(tup1)
```

```
(1, 'a', True, 'b', False)
```

[ ]:
```
# Extract one or more elemnts from a Tuple.
print(tup1[0])
print(tup1[-1])
print(tup1[0:4])
```

```
1
False
(1, 'a', True, 'b')
```

####Basic Operations on Tuple

[ ]:
```
# Finding length of Tuple

len(tup1)
```

[ ]:
```
5
```

[ ]:
```
# Concatenation of Tuples

tup_1 = (1, 2, 3, 4, 5)
tup_2 = (6, 7, 8, 9, 10)

print(tup_1 + tup_2)
```

```
(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

[ ]:
```
# Repeating Tuple Elements

print(tup_1 * 2)
```

```
(1, 2, 3, 4, 5, 1, 2, 3, 4, 5)
```

[ ]:
```
# Repeating and concatenting
print(tup_1 * 2 + tup_2)
```

```
(1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

```
[ ]: # Find maximum and minimum value

     tup_3 = tup_1 + tup_2
     print(max(tup_3))
     print(min(tup_3))
```

```
10
1
```

11.0.2 2. List in Python

List is an order collection of elements enclosed within [ ], elements can be different types (heterogeneous)

Note:
a. List are mmutable (Stored value in List can be changed)
b. All the above operations of Tuple and List are same

```
[ ]: list1 = [1, "a", True, "b", False]
     print(list1)
```

```
[1, 'a', True, 'b', False]
```

Basic Operations on List

```
[ ]: # Change an element using index
     list1[0] = "Apple"
     print(list1)
```

```
['Apple', 'a', True, 'b', False]
```

```
[ ]: # Append a new element in the list
     list1.append("Mango")
     print(list1)
```

```
['Apple', 'a', True, 'b', False, 'Mango']
```

```
[ ]: # Pop an element in the list (It works using index)

     list1.pop(3) # 3 is an index
     print(list1)
```

```
['Apple', 'a', True, False, 'Mango']
```

```
[ ]: # Remove a specific item from list
     list1.remove(True)
     print(list1)
```

```
['Apple', 'a', False, 'Mango']
```

```
[ ]: # Delete an elenment using index, del method can delete all the items of a list.
     del list1[0]
     print(list1)
```

```
['a', False, 'Mango']
```

```
[ ]: del list1
```

```
[ ]: # Clear all the elements of the list
     list1 = [1, 2, 3, 4]
     list1.clear()
     print(list1)
```

```
[]
```

```
[ ]: # Reverse the elements of a list

     list1 = [1, 2, 3, 4, 5, 6, 7, 8, 9]
     list1.reverse()
     print(list1)
```

```
[9, 8, 7, 6, 5, 4, 3, 2, 1]
```

```
[ ]: # Insert an element on a specific index (0 is index and "Apple" is an element)
     list1.insert(0, "Apple")
     print(list1)
```

```
['Apple', 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

```
[ ]: # Sort a list elements
     list_2 =  [1, 3, 4, 9, 7, 2, 5, 6, 8]
     list_2.sort()
     print(list_2)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

11.0.3 3. Dictionary

Dictionary is an unordered collection of key-values pairs enclosed in { }. **Note:** Dictionary is mutable

```
[ ]: # Creat a dictionary
     df_1 = {
           "fruit_list" : ["Apple", "Mango", "Banana"],
           "city_names" : ["Karachi", "Lahore", "Islamabad"],
           "color_names" : ["Red", "Green", "Blue"]
            }
     df_1
```

```
[ ]: {'fruit_list': ['Apple', 'Mango', 'Banana'],
      'city_names': ['Karachi', 'Lahore', 'Islamabad'],
      'color_names': ['Red', 'Green', 'Blue']}
```

Basic Operations on Dictionary

```
[ ]: # Extracting key
     dic_keys = df_1.keys()
     dic_keys
```

```
[ ]: dict_keys(['fruit_list', 'city_names', 'color_names'])
```

```
[ ]: # Extracting values
     dic_values = df_1.values()
     dic_values
```

```
[ ]: dict_values([['Apple', 'Mango', 'Banana'], ['Karachi', 'Lahore', 'Islamabad'],
     ['Red', 'Green', 'Blue']])
```

```
[ ]: # Adding a new key & value
     df_2 = {"student_names" : ["Kashif", "Asif", "Nasir" ]}
     df_2["class"] = [1, 2, 3]
     df_2
```

```
[ ]: {'student_names': ['Kashif', 'Asif', 'Nasir'], 'class': [1, 2, 3]}
```

```
[ ]: # Changing exist value
     df_2["student_names"] = ["Ammar", "Hamza", "Imran"]
```

```
[ ]: df_2
```

```
[ ]: {'student_names': ['Ammar', 'Hamza', 'Imran'], 'class': [1, 2, 3]}
```

```
[ ]: # Update/ append elements of an dictionary with another.

     data_1 = {
         "series_1" : [1, 2, 3, 4]
       }

     data_2 = {
         "series_2" : [5, 6, 7, 8]
       }

     data_1.update(data_2)
```

```
[ ]: # Pop a key and value
```

```
df_1 = {
      "fruit_list" : ["Apple", "Mango", "Banana"],
      "city_names" : ["Karachi", "Lahore", "Islamabad"],
      "color_names" : ["Red", "Green", "Blue"]
       }

df_1.pop("color_names")
df_1
```

```
[ ]: {'fruit_list': ['Apple', 'Mango', 'Banana'],
      'city_names': ['Karachi', 'Lahore', 'Islamabad']}
```

11.0.4 Set in Python

Set is unordered and unindexed collection of elements enclosed with { }, duplicates are not allowed in set.

```
[ ]: set_1 = {1, 2, 3, 4, 4} # Set don't allow duplicates.
     print(set_1)
```

```
{1, 2, 3, 4}
```

Basic Operations Sets

```
[ ]: # Update an element in set.
     set_1.add("Hello")
     print(set_1)
```

```
{1, 2, 3, 4, 'Hello'}
```

```
[ ]: # Add multiple elements in a set.
     set_1.update([6, 7, 8, 9, 10])
     print(set_1)
```

```
{1, 2, 3, 4, 6, 7, 8, 9, 10, 'Hello'}
```

```
[ ]: # Remove an element
     set_1.remove(10)
     print(set_1)
```

```
{1, 2, 3, 4, 6, 7, 8, 9, 'Hello'}
```

```
[ ]: # Uninon/ Concatenate two sets.
     set_2 = {1, 2, 3, 4, 5, 6}
     set_3 = {5, 6, 7, 8, 9, 10}
     set_2.union(set_3)
```

```
[ ]: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
[ ]: # Intersect two set (find common elements)
     set_2.intersection(set_3)
```

```
[ ]: {5, 6}
```

12 If Statement

```
[ ]: # If you got marks in exam more than or equal to 60 (Get a party)
     # else (do work hard)

     a = 60
     question = int(input("I got marks in exams: "))

     if question >= a:
         print("Get a party")
     else:
        print("Do work hard")
```

```
I got marks in exams: 65
Get a party
```

```
[ ]: # What is the greatest number

     a = 10
     b = 20
     c = 30

     if a > b & a > c:
       print("a is greatest number")
     elif b > a & b > c:
       print("b is greatest number")
     else:
       print("c is greatest number")
```

```
c is greatest number
```

```
[ ]: # If condition with Tuple

     find_data = input("Please type the name to find: ")
     tup_data = ("Umar", "Kashif", "Nasir")

     if find_data in tup_data:
       print(find_data, "is present in the data")
     else:
       print(find_data, "does not present in the data")
```

```
Please type the name to find: Umar
```

```
Umar is present in the data
```

[ ]:
```python
# If condition with List

find_data = input("Please type the name to match: ")
index_num = int(input("Please put index number: "))
list_data = ("Umar", "Kashif", "Nasir")

if find_data in list_data[index_num] == "Umar":
  print("Data has been matched")
else:
  print("Data does not match")
```

```
Please type the name to match: Umar
Please put index number: 0
Data has been matched
```

[ ]:
```python
# If statement with dictionary

num = int(input("Put a number to add: "))
dict_data = {"K1": 10, "K2":20}

if dict_data["K1"] == 10:
  dict_data["K1"] = dict_data["K1"] + num
print(dict_data)
```

```
Put a number to add: 100
{'K1': 110, 'K2': 20}
```

13 While Loop

While Loop statement is used to repeat a task multiple times.

[ ]:
```python
# Print counting from 1 to 10.

i = 1

while i <= 10:
  print(i)
  i = i + 1
```

```
1
2
3
4
```

```
5
6
7
8
9
10
```

```
[ ]: # creating table of 2 using while loop.
     i = 1
     n = 2

     while i <= 10:
       print(n,"*",i,"=",i*2)
       i = i + 1
```

```
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
2 * 6 = 12
2 * 7 = 14
2 * 8 = 16
2 * 9 = 18
2 * 10 = 20
```

```
[ ]: # Print a list using while loop.

     list_x = [1, 2, 3, 4, 5, 6, 7, 8, 10]
     i = 0

     while i < len(list_x):
       print(list_x[i])
       i = i + 1
```

```
1
2
3
4
5
6
7
8
10
```

14 For Loop

For loop is used to iterate over a sequence(string, tuple, list, dictionary..)

[ ]:
```
# Print elements of list individually.

fruit_list = ['Mango', 'Apple', 'Grapes', 'Orange']

for i in fruit_list:
  print(i)
```

```
Mango
Apple
Grapes
Orange
```

[ ]:
```
# Nested loop

item_1 = ['Orange','Black', 'White']
item_2 = ['Table', 'Chairs', 'Pen']

for i in item_1:
  for j in item_2:
    print(i,j)
```

```
Orange Table
Orange Chairs
Orange Pen
Black Table
Black Chairs
Black Pen
White Table
White Chairs
White Pen
```

15 Functions

Function is block of code which perform a specific taks.

[ ]:
```
# Create a simple function.

def hello():
  print("Hello World")

hello()
```

```
Hello World
```

[ ]:
```
# Create a function with parameters to add numbers.

def add_fun(x):
  return x + 10

add_fun(100)
```

```
[ ]: 110
```

```
[ ]: # Create a function to find value of x is even or odd

def find_even_odd(x):

  if x % 2 == 0:
    print(x, "is an even number")
  else:
    print(x, "is an odd number")
```

```
[ ]: find_even_odd(6)
find_even_odd(7)
```

```
6 is an even number
7 is an odd number
```

16 Lambda Function

A lambda function is a small anonymous function.
A lambda function can take any number of arguments, but can only have one expression.

```
[ ]: # Lambda Function is used with other function filer, map, reduce
x_function = lambda x: x + 10
x_function(2)
```

```
[ ]: 12
```

```
[ ]: # Lambda with filter function
# filter has two parameters (lambda function, num_list)

num_list = [87, 56, 90, 34, 44, 47, 27]

filtered_list = list(filter(lambda x: (x % 2 != 0), num_list))
filtered_list
```

```
[ ]: [87, 47, 27]
```

```
[ ]: # Lambda with map function
# Task is multiply the list numbers with 2, map function helps to perform a␣
 ↪task on each numbers individually.

serial_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
table_final=list(map(lambda x: (x * 2), serial_list))
table_final
```

[ ]:
```
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

[ ]:
```
# Reduce Function
# Reduce Function is used to get a final result on a sequence.

from functools import reduce
sum = reduce(lambda x,y: x+y, serial_list)
sum
```

[ ]:
```
55
```

17 Object Oriented Programming (Oops)

Creating the first Class (created function inside the class calls method)

STEPS:
a. Creating the "DataSum" class (class should be Camel Case)
b. Instantiating the "ds" object
c. Invoke methods through object

[ ]:
```
# a. Defining a class
class DataSum:
  def sum_nums (self): # self is an object mostly used in python.
    print(3+4)

# b. Instantiating object of the class
ds = DataSum()

# c. Invoke methods through object
ds.sum_nums()
```

```
7
```

17.0.1 Methods

[ ]:
```
# Use of self

class DrawCash:

  a = 2000 # created a variable outside the function

  def your_cash(self):
    # print(a) # the created variable cannot be used without the permission of␣
  ↪self, it will threw error.
    print(self.a)

dc = DrawCash()
dc.your_cash()
```

```
2000
```

[ ]:
```
# another use of self

class PayCash:

  b = 10

  def pay_cash(self):
    self.c = self.b * self.b # created a variable inside the function with the␣
  ↪permission of self as well as calculated.
    print(self.c)

pc = PayCash()
pc.pay_cash()
```

```
100
```

[ ]:
```
# use of multiple parameters with self

class MulData:
  def multi_nums (self, a, b): # use of multiple parameters with self.
    print(a * b)

md = MulData()
md.multi_nums(10 ,50)
```

```
500
```

17.0.2 Constructor

(Python supports a special type of method called constructor for initializing the instance variable of a class).

[ ]:
```
class MyClass:
  def __init__ (self): # When use __init__ it calls construction despite␣
  ↪function.
    print("Hello I am contructor")

mc = MyClass() # in __init_ there is no need to call the method.
```

```
Hello I am contructor
```

17.0.3 Inheritance

[ ]:
```
# Single level inheritance

class A:

  def print_a (self):
    print("Hello World A")

class B(A): # Single class inheritance

  def print_b (self):
    print("Hello World B")

obj_b = B() # due to Class A inherited with Class B, hence method of Class A␣
 ↪can also be called without creating its object.

obj_b.print_a()
obj_b.print_b()
```

```
Hello World A
Hello World B
```

[ ]:
```
# Multi level inheritance

class X:
  def print_x (self):
    print("This is class x")

class Y:
  def print_y (self):
    print("This is class y")

class Z(X,Y):
  def print_z (self):
    print("This is class z")

objz = Z()

objz.print_x()
objz.print_y()
objz.print_z()
```

```
This is class x
This is class y
This is class z
```