Advanced Python Programming Concepts to Solve Exercise Problems (P_VTP2)

11/16/2020

Contents

- Correspondence Index
- Class Objects
  - P_VTP2: 1,2,3,4
- Single Dimensional Arrays
  - P_VT2: 5,6,7,8,9,10,11
- JSON
- Use of JSON
- JSON Syntax
- JSON Data Types

- Conversion of JSON String to a Dictionary
  - P_VTP2: 12
- Conversion of Different Primitive Types into JSON Strings
  - P_VTP2: 15, 16
- Updating JSON String
  - P_VTP2: 17
- Data Reading From JSON file
  - P_VTP2: 13, 14
- Correspondence Between Each Topic and Related VTPs
- Conclusion

Correspondence Index

Python Advanced Grammar Concept Topic	Related Slide Number
Class Objects	6
Single Dimensional Array	10
JSON	18
Use of JSON	20
JSON Syntax	21
JSON Data Types	23
Conversion of JSON String to a Dictionary	24
Conversion of Primitive Types into JSON Strings	25

Python Advanced Grammar Concept Topic	Related Slide Number
Updating JSON Strings	27
Data Reading from JSON file	28
Correspondence Between Each Topic and Related VTPs	30
Conclusion	31

Class Objects

- Class- A user-defined **prototype for an object** that defines a set of attributes that characterize any object of the class.
- Class variable − A variable that is **shared by all instances** of a class.
- Instance − An **individual object** of a certain class.
- Method − A **special kind of function** that is defined in a class definition.
- Object − A **unique instance** of a data structure that's defined by its class. An object comprises **both** data members (class variables and instance variables) and methods.

- The first method __init__() is a special method, which is called **class constructor or initialization method** that Python calls when a new instance of this class is created.

- To create instances of a class, it is needed to call the class using **class name** and **pass in whatever arguments** its __init__ method accepts.

- The object's attributes can be accessed using the **dot operator** with object.

- #Class Constructor Function

```
class Employee:
        #Common base class for all employees
         empCount = 0
def __init__(self, name, salary):
         self.name = name
         self.salary = salary
         Employee.empCount += 1
def displayCount(self):
         print ("Total Employee %d" ,Employee.empCount)
def displayEmployee(self):
         print ("Name : ", self.name, ", Salary: ", self.salary)
```

```
#Creating Instance Object
emp1 = Employee("Zara", 2000)
emp2 = Employee("Manni", 5000)

#Accessing Object
emp1.displayEmployee()
emp2.displayEmployee()
print ("Total Employee %d“, Employee.empCount)
print (“ Employee._doc_:”, Employee._doc_)
```

Output

```
2
Common base class for all employees
```

Single Dimensional Array

- Array is a container which can hold **a fix number of items** and these items should be of **the same type**.

- The example syntax for array creation can be seen as follow:

Syntax	array_name = array(type_code, [elements])
Example-1	a = array('i', [4, 6, 2, 9])
Example-2	a = array('d', [1.5, -2.2, 3, 5.75])

This is the array creation type code for **all data types**.

Typecode	C Type	Sizes
‘b’	signed integer	1
‘B’	unsigned integer	1
‘i’	signed integer	2
‘I’	unsigned integer	2
‘l’	signed integer	4
‘L’	unsigned integer	4
‘f’	floating point	4
‘d’	double precision floating point	8
‘u’	unicode character	2

- **Accessing Array Elements Example**

```
import array
#Create an array
 a = array.array("i", [1, 2, 3, 4])
 print("Access element is: ", a[0]) >> 1
#print the items of an array
 print("Items are: ")
 for i in a:
      print(i)
```

Output

Access element is: 1

1,2,3,4

Processing the Array

Method	Description
a.append(x)	Adds an element x at the end of the existing array a
a.count(x)	Returns the numbers of occurrences of x in the array a
a.extend(x)	Appends x at the end of the array a. 'x' can be another array or an iterable object
a.index(x)	Returns the position number of the first occurrence of x in the array. Raises 'ValueError' if not found
a.insert(i, x)	Inserts x in the position i in the array

Method	Description
a.pop(x)	Removes the item x from the arry a and returns it
a.pop()	Removes last item from the array a
a.remove(x)	Removes the first occurrence of x in the array a. Raises ‘ValueError’ if not found
a.reverse()	Reverse the order of elements in the array a
a.tolist()	Converts the array ‘a’ into a list

Example program for Array

- from array import *
  #Create an array
  a = array('i', [1, 2, 3, 4, 5])
  print(a) >> array('i', [1, 2, 3, 4, 5])
  #Append 6 to an array
  a.append(6)
  print(a) # output >>> array(‘i’, [1, 2, 3, 4, 5, 6])
  #Insert 11 at position 1
  a.insert(1, 11)
  print(a) # output >>> array(‘i’, [1, 11, 2, 3, 4, 5, 6])
  #Remove 11 from the array
  a.remove(11)
  print(a) # output >>> array(‘i’, [1, 2, 3, 4, 5, 6])
  #Remove last item using pop()
  item = a.pop()
  print(a) # output >>> array(‘i’, [1, 2, 3, 4, 5])

- # Print elements of a range using Slice operation
  Sliced_array = a[1:3]
  print(Sliced_array) >>> array('i', [2,3])

  # Print elements from a pre-defined point to end
  Sliced_array = a[3:]
  print(Sliced_array) >>> array('i', [4,5])

  # Printing elements from beginning till end
  Sliced_array = a[:]
  print(Sliced_array) >>> array('i', [1,2,3,4,5])

- # updating a element in a array

  a[3] = 1

  print (a) # output >>> array('i', [1, 2, 3, 1, 5, 6])

  # using index() to print index of 1st occurrenece of 1

  print (a.index(1)) >>> 0

JSON

- JSON is a **lightweight text-based open standard** designed for **human-readable data interchange**. Conventions used by JSON are known to programmers, which include C, C++, Java, Python, Perl, etc.

  - JSON stands for **JavaScript Object Notation**.
  - The format was specified by **Douglas Crockford**.
  - It was designed for **human-readable** data interchange.
  - It has been **extended from** the JavaScript scripting laguage.
  - The filename extension is **.json**.

- JSON Internet Media type is **application/json**.
- The Uniform Type Identifier is **public.json**.
- It is also a **lightweight text-based** interchange format.
- JSON is **language independent**.
- JSON is **easy to** read and write.

Use of JSON

- It is used while **writing JavaScript based** applications that includes browser extensions and websites.
- JSON format is used for **serializing and transmitting structured data** over network connection.
- It is primarily used **to transmit data** between a server and web applications.
- Web services and APIs use JSON format to **provide public data**.
- It can be used with **modern programming languages**.

JSON Syntax

- JSON syntax is basically considered as a **subset of JavaScript syntax**. It includes the following −

  - Data is represented in **name/value pairs**.

  - Curly braces hold **objects** and each name is followed by **':'(colon)**, the name/value pairs are separated by **, (comma)**.

  - Square brackets hold **arrays** and values are separated by **,(comma)**.

- The following example shows how to **use JSON to store information** related to books based on their topic and edition.

```
{
        "id":123,
        "name":“Thomas  Edison",
        "permanent":true,
        "address":{
                        "street":“ Broadway",
                        "city":“Yangon",
                        "zipcode":12345
                },
        "phoneNumbers":[123456789, 222333444],
        "role":“Engineer"
}
```

JSON Data Types

- JSON format supports the following **data types** −

Sr.No.	Type & Description
1	**Number** double- precision floating-point format in JavaScript
2	**String** double-quoted Unicode with backslash escaping
3	**Boolean** true or false
4	**Array** an ordered sequence of values
5	**Value** it can be a string, a number, true or false, null etc
6	**Object** an unordered collection of key:value pairs
7	**Whitespace** can be used between any pair of tokens
8	**null** empty

Conversion of JSON String to a Dictionary

- To convert **from Json** string to a dictionary, **json.loads()** method can be used.

Example

```
person1 = '{"name": "Jame", "languages": ["Japanese", "Chinese"]}'
person_dict1 = json.loads(person1)
print(person_dict1)
```

Then the output is printed like this format:

```
{'name': 'Jame', 'languages': ['Japanese', 'Chinese']}
```

```
print(person_dict1['languages'])
```

Then the output is printed like this format:

```
['Japanese', 'Chinese']
```

Conversion of Different Primitive Types into JSON Strings

- To convert different primitive types into json strings, **json.dumps()** method can be used.

Example

```
print(json.dumps({"age": 30, "name": "John"}))
# for dict, the output be like {"age": 30, "name": "John"}
print(json.dumps(["apple", "bananas"]))
# for list, the output be like ["apple", "bananas"]
print(json.dumps(("apple", "bananas")))
# for tuple, the output be like ["apple", "bananas"]
print(json.dumps("hello"))
 # for string, the output be like "hello’’
```

```
print(json.dumps(42))
# for int, the output be like 42
print(json.dumps(31.76))
# for float, the output be like 31.76
print(json.dumps(True))
# for True, the output be like true
print(json.dumps(False))
# for False, the output be like false
print(json.dumps(None))
# for None, the output be like null
```

Updating JSON Strings

- To update json strings, **json.update()** method can be used.

Example

```
import json
# JSON data:
x_json = '{"name": "Smith", "languages": ["Japanese", "Korea"]}'
# python object to be appended
y_dict = {"SSN":110096}
# parsing JSON string:
z_json = json.loads(x_json)
# appending the data
z_json.update(y_dict)
# the result is a JSON string:
print(json.dumps(z_json)) # Then, the output is be like:
{"name": "Smith", "languages": ["Japanese", "Korea"], "SSN": 110096}
```

Data Reading from JSON file

- To read the content from the JSON file. Below is the implementation.

Example

Assume data.json includes as follow:

```
{ "emp_details": [
                {     "emp_name" : “Melk",
                       "email" : “melk.shubh@gmail.com",
                      "job_profile" : “Full Time"
                }
          ]
}
```

```
import json
 # JSON file Loading
f = open ('data.json',  "r")
# Reading data from JSON file
data = json.loads(f.read())
# Accessing the data inside the Json file by Iterating through the  json
    list
for i in data['emp_details']:
      print(i)
# Closing the file
f.close()
```

Then, the output is be like that:

{'emp_name': ‘Melk', 'email': ‘melk.shubh@gmail.com', 'job_profile': ‘Full Time'}

Correspondence Between Each Topic and Related VTPs

Advanced Grammar Concepts	Related Problem Number
Class Objects	1, 2, 3, 4
Arrays	5, 6, 7, 8, 9, 10, 11
Conversion of JSON String to a Dictionary	12
Conversion of Primitive Types into JSON Strings	15, 16
Updating JSON Strings	17
Data Reading from JSON file	13, 14

Index Page

Conclusion

- This slide introduces advanced concepts for Python Programming.