CEN488 Python Programming

1

Midterm Questions

1. Write a Python program where the user enters positive real numbers from the keyboard. When the user enters the first non-positive number, the number entry process ends, and the program displays the number and the average of the entered numbers.

2

Midterm Questions

2. Write a complete Python script to sketch the 2-D graph (plot) of f(x) for x=[0,6], where the following expression defines the f(x) function:

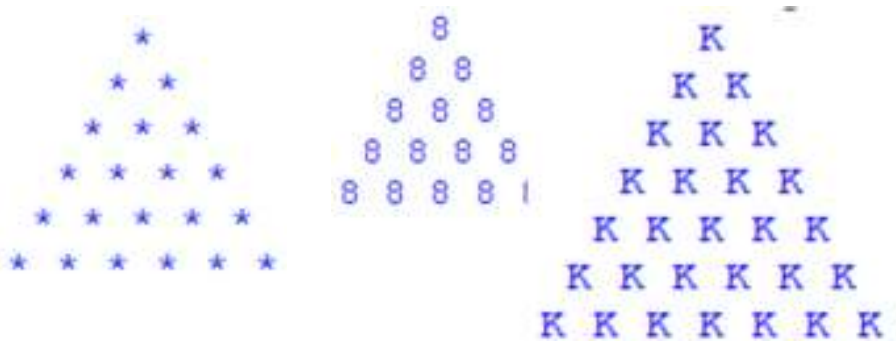
$$f(x) = 1 + \sum_{n=1}^{40} (-1)^{n-1} \frac{4}{(2n-1)\pi} \cos\left((2n-1)\pi x\right)$$

3

Midterm Questions

3. Write a Python function named "pattern" that accepts two arguments (an integer and a character) and prints the character to form a triangular shape, as shown below:

With 6 and '*' **With 5 and '8'** **With 7 and 'K'**



4

Midterm Questions

4. Write a Python function that accepts lists, where each list contains "name_surname" and CGPA of a student as its arguments. The number of students is not known. The function prints only the student's name with the highest CGPA, and it returns the list of that student. If more than one student achieved the highest grade, multiple names should be printed, and the function should return multiple lists.

5

Midterm Questions

5.Determine the output of the following code:

```
grade_pts={'A':4.00,'A-':3.7,'B+':3.30,'B':3.00,'B-':2.70,
           'C+':2.3,'C':2.0,'C-':1.7,'D+':1.3,'D':1.0,'F':0.0}
class Student:
    def __init__(self,name):
        self.name=name
        self.courses=[]
        self.cgpa=0.0

    def add_a_course(self,course,grade):
        self.courses.append([course,grade])

    def calc_gpa(self):
        total_credit=0.0
        total_points=0.0
        for crs in self.courses:
            total_credit+=crs[0][1]
            total_points+=crs[0][1]*grade_pts[crs[1]]
        self.cgpa=total_points/total_credit

course1=["MT111",4]
course2=["PS111",3]
course3=["ENG101",3]
course4=["ENG103",3]
```

6

5. Cont...

```
student1=Student("Ahmet Arkin")
student2=Student("John Seed")
student1.add_a_course(course1,'A-')
student1.add_a_course(course2,'B')
student1.calc_gpa()
student1.add_a_course(course3,'C')
student2.add_a_course(course1,'D')
student2.add_a_course(course3,'C-')
student2.add_a_course(course4,'B+')
student2.calc_gpa()
print(student1.name,student1.cgpa)
for n in student1.courses:
    print("\t",n[0][0],n[1])
print(student2.name,student2.cgpa)
for n in student2.courses:
    print("\t",n[0][0],n[1])
```

7

Midterm Questions

6. Write a python function (name it "my_swap") that accepts a list as the argument and returns a new list that contains the same elements. The new list consists of elements taken from the beginning and end of the argument, respectively: For example, the first and last elements of the argument list are the first and the second element of the new list; the second element from the beginning and the end are the third and fourth elements of the new list and so on.

8

Object Oriented Programming

Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects", which can contain data and code: data in the form of fields (often known as attributes or properties), and code, in the form of procedures (often known as methods).

Object-oriented programming (OOP) is a computer programming model that organizes software design around data, or objects, rather than functions and logic. An object can be defined as a data field that has unique attributes and behavior.

9

The structure of OOP

- Classes are user-defined data types that act as the blueprint for individual objects, attributes and methods.
- Objects are instances of a class created with specifically defined data.
- Methods are procedures (functions) that are defined inside a class that describe the behaviours of an object.
- Attributes are defined in the class template and represent the state of an object. Objects will have data stored in the attributes field. Class attributes belong to the class itself.

10

The main principles of OOP

- **Encapsulation** is defined as the wrapping up of data under a single unit. This principle states that all important information is contained inside an object and only certain information is exposed.
- **Abstraction** is a principle-based only on showing the essentials. It is similar to the generalization process.
- **Inheritance** is the mechanism that enables one class to inherit all of the state and behavior of another class
- **Polymorphism**: 'having multiple forms', Objects are designed to share behaviours to extend the functionality of the parent class.

11

Python Class Example

- **The __init__() Function (constructor)**

All classes have a function called __init__(), which is always executed when the class is being initiated.

Use the __init__() function to assign values to object properties, or other operations that are necessary to do when the object is being created:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)

print(p1.name)
print(p1.age)
```

The __init__() function is called automatically every time the class is being used to create a new object.

12

The self parameter

The self parameter is a reference to the current instance of the class, and is used to access variables that belong to the class. In fact 'self' is not a keyword in python and you may use another word instead:

```
class Person:
  def __init__(my_ref, name, age):
    my_ref.name = name
    my_ref.age = age
  def increase_age(my_ref):
    my_ref.age+=1
```

```
>>> p1=Person("Kate",25)
>>> print(p1.name,p1.age)
Kate 25
>>> p1.increase_age()
>>> print(p1.name,p1.age)
Kate 26
```

13

'del' keyword

- **'del' keyword can be used to delete an object:**

```
>>> p1=Person("Kate",25)
>>> print(p1.name,p1.age)
Kate 25
>>> del p1.age
>>> print(p1.name,p1.age)
Traceback (most recent call last):
  File "<pyshell#39>", line 1, in <module>
    print(p1.name,p1.age)
AttributeError: 'Person' object has no attribute 'age'
>>> del p1
>>> print(p1.name)
Traceback (most recent call last):
  File "<pyshell#41>", line 1, in <module>
    print(p1.name)
NameError: name 'p1' is not defined
```

14

Inheritance

- Inheritance allows us to define a class that inherits all the methods and properties from another class.
- Parent class is the class being inherited from, also called base class.
- Child class is the class that inherits from another class, also called derived class.
- For Example we may create a new class Student as a child class by using the Person as parent class:

```
class Student(Person):
    pass
```

The pass keyword is used when we do not want to add any other properties or methods to the class.

In general pass is A null statement, a statement that will do nothing

15

Example

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  pass
```

```
>>> std=Student("John","West")
>>> std.printname()
John West
```

16

Adding new attributes

- When we want to add new methods/properties to the child class we should use the __init__() function
- But, when we add the __init__() function, the child class will no longer inherit the parent's __init__() function.
- Therefore, if we need to inherit the parent's _init__() function while adding new attributes to the child we should use either the name of the parent class or we should use super() function.

17

Examples

```
class Person:
      def __init__(self, fname, lname):
            self.firstname = fname
            self.lastname = lname

      def printname(self):
            print(self.firstname, self.lastname)

class Student(Person):
    def __init__(self, fname, lname,snumber):
          self.number=snumber
```

```
>>> print(Joe.number)
2170034
```

18

```
>>> print(Joe.printname())
Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    print(Joe.printname())
  File "<pyshell#5>", line 7, in printname
    print(self.firstname, self.lastname)
AttributeError: 'Student' object has no attribute 'firstname'
```

This message is due to __init__ () function we used while defining the child class:

```
class Student(Person):
    def __init__(self, fname, lname,snumber):
            self.number=snumber
```

In order to fix this problem, we need to call __init__() function of the parent class by using one of the ways explained before as shown in the next page

19

Calling the parent's constructor 1

```
class Person:
    def __init__(self, fname, lname):
        self.firstname = fname
        self.lastname = lname
    def printname(self):
        print(self.firstname, self.lastname)

class Student(Person):
    def __init__(self, fname, lname,snumber):
        Person.__init__(self,fname,lname)
        self.number=snumber
```

```
>>> Joe=Student("Joe","Blanck",2170034)
>>> print(Joe.number)
2170034
>>> Joe.printname()
Joe Blanck
```

20

Calling the parent's constructor 2

```
class Person:
    def __init__(self, fname, lname):
        self.firstname = fname
        self.lastname = lname
    def printname(self):
        print(self.firstname, self.lastname)

class Student(Person):
    def __init__(self, fname, lname,snumber):
        super().__init__(fname,lname)
        self.number=snumber
```

```
>>> Joe=Student("Joe","Blanck",2170034)
>>> print(Joe.number)
2170034
>>> Joe.printname()
Joe Blanck
```

21

Compare child classes:

```
class Student(Person):
    def __init__(self, fname, lname,snumber):
        Person.__init__(self,fname,lname)
        self.number=snumber
```

- And

```
class Student(Person):
    def __init__(self, fname, lname,snumber):
        super().__init__(fname,lname)
        self.number=snumber
```

22

- **The super() built-in returns a proxy object (temporary object of the superclass) that allows us to access methods of the parent class.**

```
class Person:
    def __init__(self, fname, lname):
        self.firstname = fname
        self.lastname = lname
    def printname(self):
        print(self.firstname, self.lastname)

class Student(Person):
    def __init__(self, fname, lname,snumber):
        super().__init__(fname,lname)
        self.number=snumber

class EngStudent(Student):
    def __init__(self,fname,lname,snumber,CGPA):
        super().__init__(fname, lname,snumber)
        self.cgpa=CGPA
>>> st1=EngStudent("Kate","Carson",211704567,3.45)
>>> st1.printname()
Kate Carson
>>> print(st1.number,st1.cgpa)
211704567 3.45
```

23

ENCAPSULATION

- **Encapsulation is one of the fundamental concepts in object-oriented programming (OOP).**
- **Information hiding.**
- **It based on wrapping data and the methods that work on data within one unit.**
- **Rectriction on accessing variables and methods directly and can prevent the accidental modification of data (private variables)**
- **Python provides access to all the variables and methods globally**
- **In Python there are two fundamental mechanisims for encapsulation: Protected members and Private members**

24

ENCAPSULATION

- Protected Members: By convention a single underscore "_," is used in front of the name of the member to declare it is Protected
- Although the protected variable can be accessed out of the class as well as in the derived class(modified too in derived class), it is customary (convention not a rule) to not access the protected out the class body.
- Private Members cannot be accessed out of the class. the class members declared private should neither be accessed outside the class nor by any base class.
- To make class members (methods or variables) to be Private we should prefix them with double underscores '__'

25

Example

```
class Person:
    def __init__(self, fname, lname,age=25,bst="Who is"):
        self.firstname = fname
        self.lastname = lname
        self._age=age
        self.__best_friend=bst
    def printname(self):
        print(self.firstname, self.lastname)
    def print_best(self):
        print(self.__best_friend)
```

```
>>> h=Person("Ali","Saygin",23,"Test")
>>> print(h.firstname,h.lastname,h._age)
Ali Saygin 23
>>> print(h.__best_friend)
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    print(h.__best_friend)
AttributeError: 'Person' object has no attribute '__best_friend'
>>> h.print_best()
Test
```

26

Example

```
class Robot(object):
    def __init__(self):
        self.__version = 22
        self.name="Silver Line"

    def getVersion(self):
        print(self.__version)

    def setVersion(self, version):
        self.__version = version
```

Getter Method

Setter Method

27

```
>>> s1=Robot()
>>> s1.name
'Silver Line'
>>> s1.__version
Traceback (most recent call last):
  File "<pyshell#22>", line 1, in <module>
    s1.__version
AttributeError: 'Robot' object has no attribute '__vers
>>> s1.getVersion()
22
>>> s1.setVersion(23)
>>> s1.getVersion()
23
>>> s1.name="Ali"
>>> s1.name
'Ali'
```

28