CO1107
Algorithm, Data Structure & Advanced Programming

Contents

Revision of CO1102

What is Algorithm? …

- A finite sequence of steps written for an agent (e.g. Computer) to solve the problem with following properties:
  - **Definiteness**: Each step must be precisely and unambiguously specified for the agent
  - **Input**: Zero or more inputs
  - **Output**: 1 or more outputs
  - **Effectiveness**: Each step is sufficiently basic that they can be done exactly and in a finite length of time by the agent
  - **Finiteness**: must always terminate after a finite number of steps

Data Structure

- In computer terms, a data structure is a specific way to store and organize data in a computer's memory so that these data can be used efficiently later.
- Data structures are particular ways of storing data to make some operation easier or more efficient.
- Data structures are suited to solve certain problems, and they are often associated with algorithms.

Categories of Data Structure

Two major types of data structures:

- Linear Data Structure
- Non Linear Data Structure

Linear Data Structure

- The data structure where data items are organized sequentially or linearly where data elements attached one after another is called linear data structure.
- Data elements in a liner data structure are traversed one after the other and only one element can be directly reached while traversing.
- All the data items in linear data structure can be traversed in single run.

Linear Data Structure

- There are two techniques of representing such linear structure within memory.
- The first way is to provide the linear relationships among all the elements represented using linear memory location. These linear structures are termed as arrays.
- The second technique is to provide the linear relationship among all the elements represented by using the concept of pointers or links. These linear structures are termed as linked lists.

Common Example of Linear Data Structure

- Arrays
- Queues
- Stacks
- Linked Lists

Non Linear Data Structure

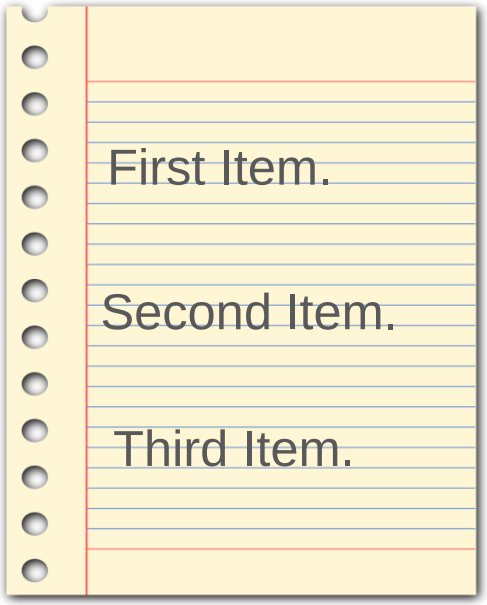
- These are the data structures in which there is no sequential linking of data elements.
- Any pair or group of data elements can be linked to each other and can be accessed without a strict sequence.
- All the data elements in non linear data structure can not be traversed in single run.

  - Binary Tree
  - Heap
  - Graph

Python Specific Data Structure

- Python comes with a general set of built in data structures:
  - lists
  - tuples
  - string
  - dictionaries
  - sets
  - others...

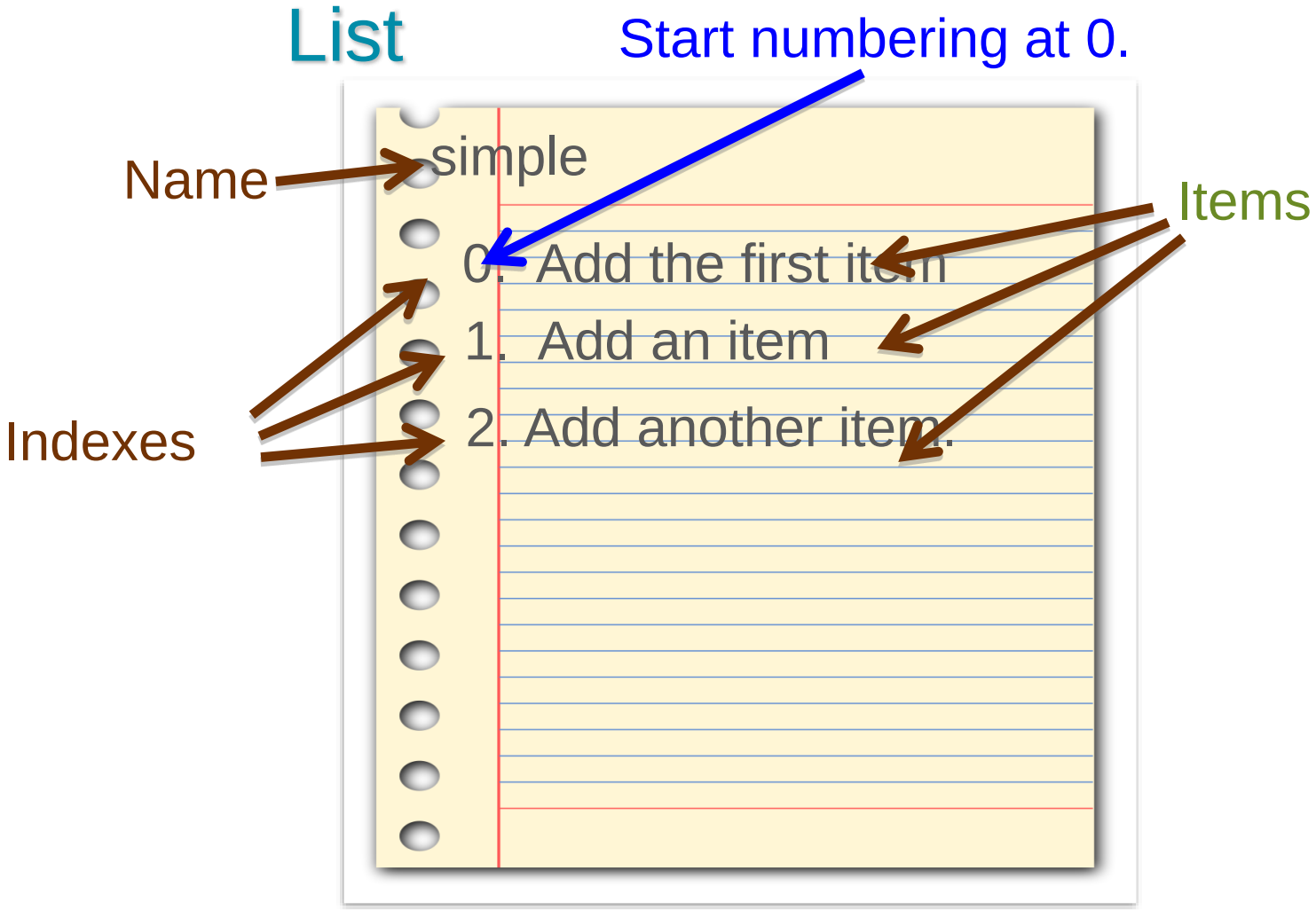
List: is a kind of collection that can hold many values in a single variable



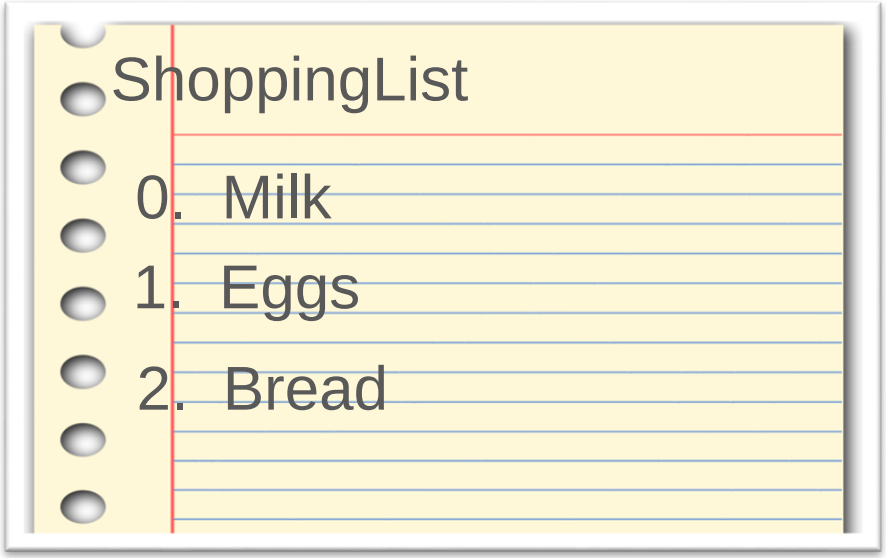
Tables

Team	P	W	D	L	F	A	Pts
Manchester United	6	4	0	2	10	5	12
Celtic	6	3	0	3	8	9	9
Benfica	6	2	1	3	7	8	7
FC Copenhagen	6	2	1	2	5	8	7

List



Notation




```
>>> ShoppingList= ['Milk', 'Eggs']
>>> ShoppingList
['Milk', 'Eggs']
>>> ShoppingList.append('Bread')
>>> ShoppingList
['Milk', 'Eggs', 'Bread']
```

Tuples

- Tuples are ordered, immutable collections of elements.

- The only difference between a tuple and a list is that once a tuple has been made, it can't be changed!
- Making a tuple:
  - a = (1, 2, 3)
- Accessing a tuple:
  - someVar = a[0]
- The syntax for access is exactly like a list. However, you can't reassign things.

What Is A Set?

- A **set** is a **collection of objects**
- The objects in a set can be anything: numbers, fish, people, cars, other sets…
- Each object in a set is called a **member** or an **element** of the set
- The elements of a set are **not ordered**
- A set can have **any number** of elements
- All of the elements of a set must be **different**

Basic Set Operations

- There are four basic operations on sets:

1) **Add** an element in a set
2) **Remove** an element from a set
3) Query whether a set **contains** a given element
4) Supply the **size** of a set

Examples of Sets

- Create Set

```
>>> theSet = { "Jan", "Feb", "Mar"}
>>> print(theSet)
    {'Jan', 'Feb', 'Mar'}
```

- Access Items

```
>>> for item in theSet:
        print(item)
    Jan
    Feb
    Mar
```

Examples of Sets

- **Once a set is created, you cannot change its items, but you can add new items.**
- Add Item

```
>>> theSet = { "Jan", "Feb", "Mar"}
```

```
>>> theSet.add(“Apr”)
```

- Add Multiple Items

```
theSet.update(["May", "Aug", "Dec"])
```

Examples of Sets

- Removing an element
  - Using remove(e)
  - Using discard(e)

```
>>> theSet = { "Jan", "Feb", "Mar"}

>>> theSet.remove("Apr")
```

Error Message since the element does not exist

```
>>> theSet.discard("Apr")
```

No error message

```
>>> theSet.remove(“Jan")

>>> theSet

{'Feb', 'Mar'}
```

Dictionaries

- A dictionary is a collection which is unordered, changeable and indexed. In Python dictionaries are written with curly brackets, and they have keys and values.
- They can be created via a comma-separated list of

  `key: value` pairs within curly braces.

```
>>> thisdict =    {
  "One": 1,
  "Two": 2,
  "Three": 3}
>>> print(thisdict)
{'One': 1, 'Two': 2, 'Three': 3}
```

Accessing Items in Dictionaries

- By referring to its key name:

```
>>> x = thisdict["One"]
>>> print(x)
1
```

- Using a get method

```
>>> x = thisdict.get(“One”)
>>> print(x)
1
```

Updating Dictionary

```
>>> thisdict["One"] = 11
>>> thisdict
{'One': 11, 'Two': 2, 'Three': 3}
```

Adding new element

```
>>> thisdict['Four']=4
>>> thisdict
{'One': 11, 'Two': 2, 'Three': 3, 'Four': 4}
```

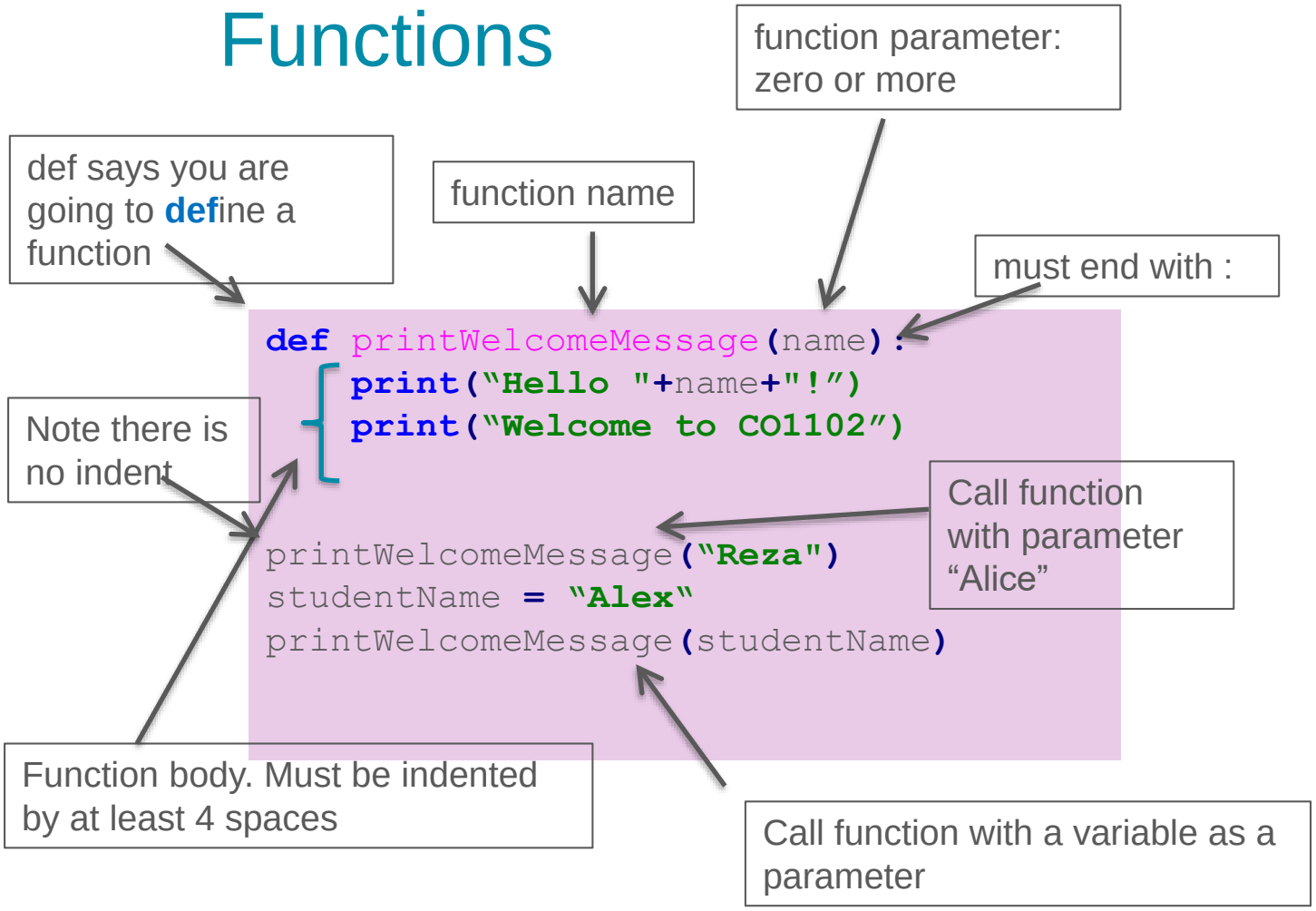
Function

Functions and Algorithms

- Functions correspond to algorithms
- Algorithms have:
  - Names
  - Input: Zero or more
  - Output: One or more

- Functions have names
- Functions may or may not have input.
- Functions may either
  - Print a value or values
  - Change an input value or values
  - Return a value or values

Functions



Functions

```
import math
# function that prints area of a circle with given
radius
def printCircleArea(radius):
    area = math.pi*radius*radius
    print("Area of the circle is",area)

printCircleArea(10)
r = 20
printCircleArea(r)
```

Functions

```
import math
def getCircleArea(radius):
    area = math.pi*radius*radius
    return area


circleArea = getCircleArea(10)
print("Area of the circle is", circleArea)
```

Value is returned to the place of call

A function may also return a value

Functions

```
def getCylinderVolume(radius,height):
    return height*getCircleArea(radius)

print(getCylinderVolume(20,4))
```

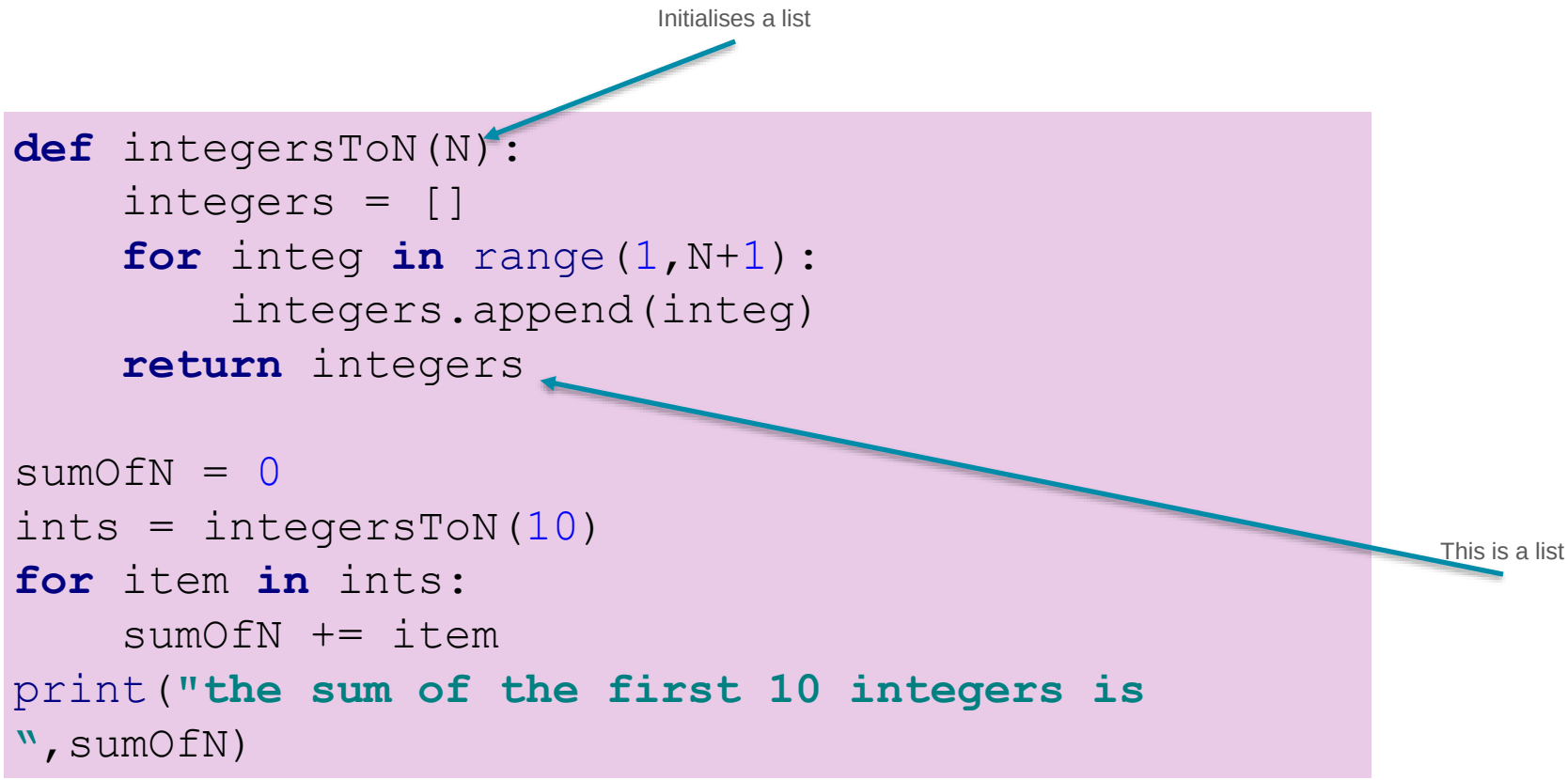
```
import math
def getCircleArea(radius):
    area = math.pi*radius*radius
    return area


circleArea = getCircleArea(10)
print("Area of the circle is", circleArea)
```

Class Activity

- Write a program to get 3 numbers from user, using the function **Sum_Num** , add up the first 2 numbers, and then multiply its result with the 3rd number by using another function called **Product.**

You are not limited to returning numbers




```
def integersToN(N):
    integers = []
    for integ in range(1,N+1):
        integers.append(integ)
    return integers

sumOfN = 0
ints = integersToN(10)
for item in ints:
    sumOfN += item
print("the sum of the first 10 integers is
",sumOfN)
```

Calculate the smallest number in a list

```
def smallest(aList):
    min=aList[0]
    for i in range(1,len(aList)):
        if aList[i]<min:
            min=aList[i]
    return min


aList=[10,7,6,3,5,9]
print(smallest(aList))
```

```
def add_evenNum(aList):
    sum=0
    for i in range(len(aList)):
        if aList[i]%2==0:
            sum=sum+aList[i]
    return sum

def maximum_number(aList):
    max=aList[0]
    for i in range(len(aList)):
        if aList[i]>max:
            max=aList[i]
    return max

def product(res,max):
    print("The product of the 2 given numbers is: ",res*max)

aList=[]
for i in range(5):
    num=int(input("Enter a number: "))
    aList.append(num)

result=add_evenNum(aList)
maximum=maximum_number(aList)
product(result,maximum)
```

Quick Introduction to Object Oriented Programming, Object and Classes

Constructor

- A constructor is a special type of method (function) which is used to initialize the instance members of the class.
- There are two types of constructor:
  - Default
  - Parametrized

Default Constructor

```
class car():
    def __init__(self):
        self.Color="White"
        self.Speed= 200
        self.Model= Benz
        self.Transmission="Auto"
```

Parameterized Constructor

```
class car():
    def __init__(self, col, speed, model, transmission):
        self.Color=col
        self.Speed=speed
        self.Model=model
        self.Transmission=transmission
```

Class Exercise

- Implement the following Person class with two attributes: Name and Age
- Create a new class, name it Person
- Add default constructor
- Add parameterized constructor with two attributes
- Add mutator and accessor methods for each attribute
- Add a print function and also test if each mutator and accessor methods works fine.

```
class person(object):
    def __init__(self):
        self.Name=""
        self.Age=30

    def __init__(self, name,age):
        self.Name=name
        self.Age=age
```

```
def getName(self):
        return self.Name
    def setName(self, name):
        self.Name=name
    def getAge(self):
        return self.Age
    def setAge(self,age):
        self.Age=age

    def __str__(self):
        return 'the person name is: '+
str(self.Name) + ' and the age is: '+
str(self.Age)

p1=person("Reza", 40)
print(p1.getName())
print(p1)
```