**Lecture 2: Input and output, variables, data type, operators and expressions**

|  |  |  |
| --- | --- | --- |
| **Data type** | **Description** | **Code example** |
| Integer | 32 bits long. From -232 to 232 – 1 |  |
| Long integers | Unlimited precision. Subject to memory limitations |  |
| Floating Point Numbers | Or Double-precision number. Use 64 bits. |  |
| Boolean | True or false | var = False |
| Complex Number | Represented by float types in Python | var = 2 + 3j |
| Strings | Sequences of Unicode characters  Single and double quotes can be used.  Triple quotes for strings with multiple line | 'Hello' / "Hello"  '''Hello  Rain''' |
| Lists | Ordered sequences of values | var = [1, 'b', 3.0] |
| Tuples | Ordered, immutable sequences of values | var = (1, 'b', 3.0) |
| Sets | Unordered collections of values | var = {1, 'b', 3.0} |
| Dictionaries | Unordered collections of key-value pairs | var = {1:'a', 'b':2.0} |

* Leading zeros are used for base 2, 8 and 16 numbers. 0b for binary, 0o for octal and 0x for hexadecimal.
* **Literal** is number or string that appears directly in a program. For example, 10 is integer literal.
* **Keywords** are identifiers that Python reserves for special use.
* **Comments** begin by #.
* **Continuation Line** is used to join two adjacent physical line (line that you see) into a logical line (a single statement) with \. Or, open parenthesis is not closed, Phyton joins adjacent line into one logical line.

|  |  |  |
| --- | --- | --- |
| Printing | print(["message"][variable list]) | print("length: ", l) |
| To suppress printing of newline character, end with end=' ' | print("Hello!", end=" ') |
| Concatenate string by + or , | print('Hello!'+'Rain') |
| Format codes for substituting values with %   |  |  | | --- | --- | | **Format code** | **Usage** | | %s | Displays in string format | | %d | Displays in decimal format | | %e | Displays in exponential format | | %f | Displays in floating-point format | | %o | Displays in octal (base 8) format | | %x | Displays in hexadecimal format | | %c | Displays ASCII code | | print ("Length is %d" %(l) |
| Input | String input | s = input('Enter string :') |
| Integer input | i = int(input('Integer :')) |
| Float input | f = float(input(Float :')) |

**Lecture 3: Arithmetic operations, bitwise operations and complex numbers**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **Operation** | **Description** | | x + y | Addition | | x - y | Subtraction | | x \* y | Multiplication | | x / y | Division | | x // y | Truncating division | | x \*\* y | Exponentiation | | x % y | Modulo operator | | –x | Unary minus | | +x | Unary plus | | | * / return an integer if both operands are integers. * // ignoring the remainder   **Multiple assignment statement**  p,q,r=10,20,30  sum, avg=p+q+r,(p+q+r)/3  **Finding data type**  Use type() function.  a = 10  print(type(a)) 🡪 <class 'int'> |
| |  |  | | --- | --- | | **Escape Character** | **Description** | | \a | Bell (beep) | | \b | Backspace | | \f | Form feed | | \n | Newline | | \r | Carriage return | | \t | Tab | | \v | Vertical tab | | \\ | Literal backslash | | \' | Single quote | | \" | Double quote | | **Displaying octal and hexa values**  c=19  print ('19 octal is %o and hex is %x' %(c,c))  d=oct(c)  e=hex(c)  print ('19 octal is', d, 'and hexa is', e)  Output:  19 octal is 23 and hex is 13  19 octal is 0o23 and hexa is 0x13  **Complex Number**  a = 3.0 + 1.2j  a.real 🡪 3.0 a.imag 🡪 1.2 | | |
| |  |  | | --- | --- | | **Bitwise Operation** | **Description** | | x << y | Binary shift left | | x >> y | Binary shift right | | x & y | Bitwise AND | | x | y | Bitwise AND | | x ^ y | Bitwise exclusive AND | | ~ x | Bitwise inversion | | | |  |  | | --- | --- | | a=10  b=7  c=a & b 🡪 2  d=a ^ b 🡪 13  e=a | b 🡪 15  g=a<<2 🡪 40  h=a>>1 🡪 5 | a=1010 in binary  b=0111 in binary  c=0010 (2 in decimal)  d=1101 (13 in decimal)  e=1111 (15 in decimal)  g=101000 (40 in decimal)  h=0101 (5 in decimal) | | |

**Lecture 4: Logical operators and loops**

|  |  |
| --- | --- |
| **if…else statement** | **Comparison operator** |
| if (logical):  statement(s)  else:  statement(s) | |  |  | | --- | --- | | **Operator** | **Meaning** | | < | Less than | | > | Greater than | | <= | Less than or equal to | | >= | Greater than or equal to | | == | Equal to | | != | Not equal to | |
| **if-elif-else statement** |
| if (logical):  statement(s)  elif (logical 1):  statement(s)  [elif (logical n):  statement(s)]  else:  statement(s) |
| **Logical operator** |
| |  |  | | --- | --- | | **Operator** | **Meaning** | | and | Return true if all logical expressions are true | | or | Return true if any of logical expression is true | | not | Negated the logical operator. Turn true to false and vice versa. | |
| **while loop** |
| while expression:  statement1  statement2  statement3 |
| **for loop** | **Membership operators** |
| for iterating\_var in sequence:  statement1  statement2  statement3 | |  |  | | --- | --- | | **Operator** | **Meaning** | | in | Return true if it finds the specified variable in given sequence otherwise return false | | not in | Return true if does not find in given sequence. | |
| **Break statement** | **range() function** |
| Terminates and exits from current loop | range(x) returns a list whose items are consecutive integers from 0 (inclusive) to x (exclusive)  range(x,y) returns a list whose items consecutive integers from x to y.  range(x,y,step) returns a list from x to y with difference between each value of step |
| **Continue statement** |
| Stops execution of current iteration by skipping rest of the loop and continuing to execute loop with next iterative value |
| **Pass statement** | **Chaining Comparison Operators** |
| Indicate empty block of statements | x<=y and y<=z is the same as x<= y <=z |

|  |  |  |  |
| --- | --- | --- | --- |
| k=1  while k <=10 :  print (k)  k=k+1 | k=1  while 1 :  print (k)  k=k+1  if(k>10):  break | k=1  while k <=10 :  if k==7:  k+=1  continue  print (k)  k=k+1 | k=1  while k <=10 :  if k==7:  pass  else:  print (k)  k+=1 |
| 1  2  3  4  5  6  7  8  9  10 | 1  2  3  4  5  6  7  8  9  10 | 1  2  3  4  5  6  8  9  10 | 1  2  3  4  5  6  8  9  10 |
| print ("Odd numbers between 1 and 10 are:")  for i in range(1,11,2):  print (i) | | * ab in abcde—Returns true because the string ab is found in the string abcde. * 2 in (10,3,5,2,1)—Returns true because the value 2 exists in the tuple. * bob not in ab—Returns true because the string bob is not found in the string ab. | |
| 1  3  5  7  9 | |

**Lecture 5: Sequences, strings and sets**

* Tuples and strings are immutable – they cannot be modified after they are created.
* List is mutable, you can append elements, remove existing elements or rearrange elements.

|  |  |
| --- | --- |
| **Operator** | **Description** |
| + | Concatenate sequences to make longer sequences |
| \* | Repeat the sequence several times |
| [ ] | Fetched a particular element from sequence (indexing) or a subset of elements from sequence (slicing) |

String Methods and functions

|  |  |
| --- | --- |
| **Functions** | **Usage** |
| len() | Length of string |
| isdigit() | Check if string is numeric |
| count(s,[start],[end]) | Returns number of occurrences of substring s in a string. If start and end is specified, s is searched and counted within the range |
| find(s,[start],[end]) | Returns lowest index in a string where s is found. Returns -1 if not found. |
| index(s,[start],[end]) | Returns lowest index in a string where s is found. ValueError if not found. |
| rfind(s,[start],[end]) | Returns highest index in a string where s is found. Returns -1 if not found. |
| startswith(p,[start],[end]) | Returns true if string starts with p. p can be single string or sequence of individual strings. |
| endswith(s,[start],[end]) | Returns true if string ends with s. s can be single string or sequence of individual strings. |

|  |  |
| --- | --- |
| **One-Dimensional Array** | **Two-Dimensional Array** |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | 8 | 3 | 1 | 6 | 2 | | p[0] | p[1] | p[2] | p[3] | p[4] | | |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 | p[0][0] | p[0][1] | p[0][2] | | 1 | p[1][0] | p[1][1] | p[1][2] | | 2 | p[2][0] | p[2][1] | p[2][2] | |

* **List** is a collection of elements, which might include other lists.

Example: ["John", "Kelly", 1, 2, [Sugar, Butter, 10]]

* First element of list is at index 0. Last element is at index -1.
* len() function returns length as an index location of the last element plus one.
* **List slicing**: Slice a list into parts to get desired element.

Syntax: list[first\_index:following\_index]

* **Tuples** can contain elements of any type.
* **Dictionary**: combination of key/value pairs in which every key has to be unique.

Syntax: d = {key1 : value1, key2 : value2}

* Dictionaries are mutable, which means it can be modified.
* Dictionary keys are case sensitive and immutable because it is associates with hash.
* Keys can be strings, integers or others.
* **Set** is a collection of certain values.

|  |  |
| --- | --- |
| **Operator** | **Description** |
| | | Union operation. An element appears in the union if it exists in one set or the other.  S1=set([3,5,6,10,11,100])  S2=set([1,3,5,6,11,15])  S1 | S2 🡪 set([1,3,5,6,10,11,15,100]) |
| & | Intersection operation. An element appears in intersection if the element appear in both sets.  S1 & S2 🡪 set([3,5,6,11]) |
| - | Difference operation. All elements that are in the left set but not in the right set will appear.  S1 - S2 🡪 set([10,100]) |

**Lecture 6: Functions**

* Functions are treated as object, so you can pass a function as an argument to another function.
* Function can return another function
* Syntax:

def function-name(parameters):

statement(s)

* Default value parameters: parameters listed in the function definition may be mandatory or optional.
* Keyword argument: naming the parameters when passing argument if supplying few parameters.

Example:

def volume(l, b=5, h=10):

...

volume(h=7, l=2)

|  |  |
| --- | --- |
| **Global variable** | **Local Variable** |
| def compute():  global x  print ("x in compute function is", x)  x += 5  return None  def dispvalue():  global x  print ("x in dispvalue function is", x)  x-=2  return None  x=0  compute()  dispvalue()  compute() | def compute(x):  x += 5  print ("x in function is", x)  return None  x=10  compute(x)  print ("x is still", x) |
| x in compute function is 0  x in dispvalue function is 5  x in compute function is 3 | x in function is 15  x is still 10 |

* **Global variables** are not bound to any particular function and can be accessed within, outside or any function. Changes made to global variable will be visible. Syntax: global identifiers

|  |  |  |
| --- | --- | --- |
| **filter()** | **map()** | **reduce()** |
| Returns a sequence consisting of those elements which the included function returns true, those that satisfy the criteria given in the specified function. | Calls the included function for each of the elements in the sequence and returns a list of the returned values. | Returns a single value that is produced by calling the function on the first two elements of the sequence. |
| def even(x):  return x % 2 ==0  evens=filter(even,range(1,11))  print(list(evens))  **Output:**  [2,4,6,8,10] | def squ(x):  return x\*x  sqr=map(squ,range(1, 11))  print(list(sqr))  **Output:**  [1, 4, 9, 16, 25, 36, 49, 64, 81, 100] | import functools  def add(x,y):  return x+y  r=functools.reduce(add, range(1, 11))  print(r)  **Output:**  55 |
|  | k=map(int,[5,10.8,20, 6.6])  print(list(k))  **Output:**  [5, 10, 20, 6] |  |

**Function Attributes**. Use to get more information about a function.

|  |  |
| --- | --- |
| **Attribute** | **Description** |
| functionname.\_\_doc\_\_ | Represents the docstring from the first line of the function’s body. |
| functionname.\_\_name\_\_ | Represents the function name. |
| functionname.\_\_module\_\_ | Represents the name of the module in which the function is defined. |
| functionname.\_\_defaults\_\_ | Represents the tuple with default values to be assigned to the default arguments of the function. |
| functionname.\_\_code\_\_ | Represents the actual code object, the statements in the body of the function. |
| functionname.\_\_dict\_\_ | Represents the dictionary that defines the local namespace for the attributes of the function. |

Example:

|  |
| --- |
| def sum(a, b=5):  "Adds the two numbers"  return a + b  sum.version= "1.0"  sum.author= "bintu"  k=sum(10,20)  print ('Sum is', k)  print('The documentation string is', sum.\_\_doc\_\_)  print('The function name is', sum.\_\_name\_\_)  print('The default values of the function are', sum.\_\_defaults\_\_)  print('The code object of the function is', sum.\_\_code\_\_)  print('The dictionary of the function is', sum.\_\_dict\_\_)  **Output:**  Sum is 30  The documentation string is Adds the two numbers  The function name is sum  The default values of the function are (5,)  The code object of the function is <code object sum at 0x00F11250, file "D:\python\funcattrib.py", line 1>  The dictionary of the function is {'version': '1.0', 'author': 'bintu'} |

**Lecture 7: Recursions and iterations**

* **Recursion** is said to occur when a function calls itself. An exit condition must be included in function to avoid infinite loop.
* **Iterators** are used for looping through collections of data. It has a next() method to get each value in the sequence.
* iter() method is used to get an iterator object. iter(object) calls object’s \_\_iter\_\_ object.

|  |
| --- |
| names=['John', 'Kelly', 'Caroline', 'Paula']  i = iter(names)  print (i.\_\_next\_\_()) 🡪 John  print (i.\_\_next\_\_()) 🡪 Kelly |

* **Generator** is a function that creates an iterator. It uses yield to get the next value. yield is used in body of generator function.
* yield converts function into generator function. When generator function is called, it returns an iterator, known as generator iterator or generator.
* The body of generator function is executed by calling \_\_next\_\_() method until raises exception.

|  |
| --- |
| def fruits(seq):  for fruit in seq:  yield '%s' % fruit  f=fruits(['Apple', 'Orange', 'Mango', 'Banana' ])  print (f.\_\_next\_\_()) 🡪 Apple  print (f.\_\_next\_\_()) 🡪 Orange |

* **Generator** **expression** is an expression in parentheses that creates an iterator object.

|  |
| --- |
| def squarenum(x):  return x\*x  iteratorobj = (squarenum(x) for x in range(6))  print (iteratorobj.\_\_next\_\_()) 🡪 0  print (iteratorobj.\_\_next\_\_()) 🡪 1 |

* **Module** is a file consisting of a few functions and variables used for a particular task. Use import statement to import a module.

**First way:** import calendar. To use prcal() function, use calendar.prcal()

**Second way:** from calendar import prcal. To use prcal() function, use prcal()

**Third way:** from calender import \*. To use prcal() function, use prcal()

**Fourth way:** import calendar as cal. To use prcal() function, use cal.prcal()

* **Math Module** (import math)

|  |  |
| --- | --- |
| **Function** | **Usage** |
| math.pi | Return value of pi |
| math.e | Return value of e |
| math.ceil(x) | Display next larger whole number |
| math.floor(x) | Display next smaller whole number |

* **dir()** function is used to list the identifiers defined by a module. Identifiers are the functions, classes and variable defined in that module.
* **Command-line arguments** are used to pass arguments to a program while running it. Each command-line argument that you pass will stored in sys.argv variable.
* sys.argv always has a length of at least 1. Index 0 is the name of Python program running.
* sys.path returns path of the Python.

|  |  |
| --- | --- |
| **Sum of 10 number using recursion** | **Calculate factorial of 5 using recursion** |
| def addseq(x):  if x == 1: return 1  else: return x + addseq(x-1)  print ('The sum is', addseq(10)) | def fact(x):  if x == 1: return 1  else: return x \* fact(x-1)  print ('Factorial of 5 is', fact(5)) |

**Lecture 8: Classes**

* Syntax:

class classname(base-classes):

statement(s)

* Base classes are optional. To create a class without bases, omit base-classes placing colon after classname.
* Attribute of a class object, bind a value to an identifier within the class body
* Defining functions in a class, functions defined is known as **method**.
* Method has a mandatory first parameter **self** that refers to the instance on which you call the method.
* Class variable: data member that is outside of any method of class. All instances of class share the variable and changes will be seen to other instances.
* Instance variable: variable defined inside a method that belong to the current instance only. Changes made to the variable don’t affect the instance variables of other instances.

|  |  |
| --- | --- |
| **Method** | **Description** |
| \_\_init\_\_() | Constructor. Arguments may or may not be passed to the method. self refers to newly created instance, while in other method, it refers to the instance whose method was called. It must not return a value, or TypeError. |
| \_\_str\_\_() | Print statements to display string representation of an instance. |

* **Class Methods** has no self argument and receives a class as its first argument. In a class method, the class on which it is called is passed to it as the first argument. Class method can be called directly through class object without instantiating the class.

|  |
| --- |
| class book:  price=100  @classmethod  def display(cls):  print (cls.price)  def show(self,x):  self.price=x  print (self.price)  b=book()  c=book()  book.display() 🡪 100  b.display() 🡪 100  b.show(200) 🡪 200  c.show(300) 🡪 300 |

* **Static method** is an ordinary function that is built using @staticmethod decorator and binds its result to a class attribute.

|  |
| --- |
| staticlassmethod.py  class product:  count = 0  def \_\_init\_\_(self, name):  self.name=name  Output:  Static method - The product count is: 2  Class info: <class '\_\_main\_\_.product'>  Class method - The product count is: 2  product.count += 1  @staticmethod  def prodstatcount():  return product.count  @classmethod  def prodclasscount(cls):  print('Class info: ', cls)  print ('Class method - The product count is: ', cls.count)  p1=product('Camera')  p2=product('Cell')  print('Static method - The product count is: ', product.prodstatcount())  p2.prodclasscount() |

|  |  |
| --- | --- |
| **Class Method** | **Static Method** |
| Has cls, self parameter | No cls, self parameter |
| Automatically inherited by ay child classes | Do not inherited by any child classes |

* Assigning an instance to another results in creation of a new instance if it doesn’t exists.

|  |
| --- |
| class rect:  Output:  Area of rectangle is 40  Area of rectangle is 40  def \_\_init\_\_(self, x,y):  self.l = x  self.b = y  def rectarea(self):  return self.l \* self.b  r=rect(5,8)  s=r  print ("Area of rectangle is ", r.rectarea())  print ("Area of rectangle is ", s.rectarea()) |

**Lecture 9: Classes – Inheritance**

|  |  |  |
| --- | --- | --- |
| **Single Inheritance** | **Access Control Specifiers** | **Method overriding** |
| One class is derived from another single class. Derived class has to identify the class from which it is derived. | Defines the visibility of the members of the class.  Public member: Accessed from inside and outside of class.  Private member: Cannot be accessed from outside the class. Preceded by \_\_. To access outside the class, use r.\_rect\_\_l | If in a derived class you have a member function with the same signature as that of the base class, then you say that the member function of the derived class is overriding the member function of the base class. |
| from \_\_future\_\_ import division  class rect:  def \_\_init\_\_(self):  self.l = 8  self.b = 5  def rectarea(self):  return self.l \* self.b  class triangle(rect):  def \_\_init\_\_(self):  rect.\_\_init\_\_(self)  self.x = 17  self.y = 13  def trigarea(self):  return 1/2\*self.x \* self.y  r=triangle()  print ("Area of rectangle is ",\  r.rectarea())  print ("Area of triangle is ",\  r.trigarea())  **Output:**  Area of rectangle is 40  Area of triangle is 110.5 | class rect:  def \_\_init\_\_(self, x,y):  self.\_\_l = x  self.\_\_b = y  def rectarea(self):  return self.\_\_l\*self.\_\_b  r=rect(5,8)  print ("Area of rectangle is ", r.rectarea())  print ("Area of rectangle is ", r.\_rect\_\_l\* r.\_rect\_\_b)  **Output:**  Area of rectangle is 40  Area of rectangle is 40 | from \_\_future\_\_ import division  class rect:  def \_\_init\_\_(self):  self.l = 8  self.b = 5  def area(self):  return self.l \* self.b  class triangle(rect):  def \_\_init\_\_(self):  rect.\_\_init\_\_(self)  self.x = 17  self.y = 13  def area(self):  return 1/2\*self.x \* self.y  r=triangle()  print ("Area of triangle is ", r.area())  **Output:**  Area of triangle is 110.5 |

|  |  |
| --- | --- |
| **Accessing methods of a base class from a derived class** | **Multilevel Inheriance** |
| from \_\_future\_\_ import division  class rect:  def \_\_init\_\_(self):  self.l = 8  self.b = 5  def area(self):  print("Area of rectangle is ",  self.l\*self.b)  class triangle(rect):  def \_\_init\_\_(self):  rect.\_\_init\_\_(self)  self.x = 17  self.y = 13  def area(self):  rect.area(self)  print("Area of triangle is ",  1/2\*self.x\*self.y)  r=triangle()  r.area()  Output:  Area of rectangle is 40  Area of triangle is 110.5 | from \_\_future\_\_ import division  class worker:  def \_\_init\_\_(self, c, n, s):  self.code = c  self.name= n  self.salary = s  def showworker(self):  print ("Code is ", self.code)  print ("Name is ", self.name)  print ("Salary is ", self.salary)  class officer(worker):  def \_\_init\_\_(self, c,n,s):  worker.\_\_init\_\_(self,c,n,s)  self.hra = s\*60/100  def showofficer(self):  worker.showworker(self)  print("HRA-House Rent Allowance is", self.hra)  class manager(officer):  def \_\_init\_\_(self, c,n,s):  officer.\_\_init\_\_(self,c,n,s)  self.da = s\*98/100  def showmanager(self):  officer.showofficer(self)  print ("DA - Dearness Allowance is ", self.da)  w=worker(101, 'John' , 2000)  o=officer(102, 'David', 4000)  m=manager(103, 'Ben' , 5000)  print ("Information of worker is ")  w.showworker()  print ("\nInformation of officer is ")  o.showofficer()  print ("\nInformation of manager is ")  m.showmanager() |
| **Multiple Inheritance: A class is derived from more than one base class.** | **Operator overloading: apply arithmetic operators to a class instance to perform desired operation** |
| from \_\_future\_\_ import division  class student:  def \_\_init\_\_(self, r, n):  self.roll = r  self.name= n  def showstudent(self):  print ("Roll : ", self.roll)  print ("Name is ", self.name)  class science:  def \_\_init\_\_(self, p,c):  self.physics = p  self.chemistry=c  def showscience(self):  print ("Physics marks: ", self.physics)  print ("Chemistry marks: ",self.chemistry)  class results(student,science):  def \_\_init\_\_(self, r,n,p,c):  student.\_\_init\_\_(self,r,n)  science.\_\_init\_\_(self,p,c)  self.total = self.physics+self.chemistry  self.percentage=self.total/200\*100  def showresults(self):  student.showstudent(self)  science.showscience(self)  print("Total marks : ", self.total)  print("% marks: ", self.percentage)  s=results(101, 'David', 65, 75)  print ("Result of student is ")  s.showresults() | **Overload +**  class rect:  def \_\_init\_\_(self, x,y):  self.l = x  self.b = y  def \_\_str\_\_(self):  return 'Length is %d, Breadth is %d' %(self.l, self.b)  def \_\_add\_\_(self, other):  return rect(self.l+ other.l, self.b+other.b)  def rectarea(self):  return self.l \* self.b  r1=rect(5,8)  r2=rect(10,20)  r3=r1+r2  print (r3)  print ("Area of rectangle is ", r3.rectarea())  **Overload ==**  class rect:  def \_\_init\_\_(self, x,y):  self.l = x  self.b = y  def \_\_str\_\_(self):  return 'Length is %d, Breadth is %d'  % (self.l, self.b)  def \_\_eq\_\_(self, other):  return((self.l==other.l)and(self.b==other.b))  def rectarea(self):  return self.l \* self.b  r1=rect(5,8)  r2=rect(10,20)  if r1==r2 :  print('The two instances are equal')  else:  print('The two instances are not equal') |

Polymorphism

|  |
| --- |
| class book:  def \_\_init\_\_(self,x):  self.price = x  class stockist(book):  def \_\_init\_\_(self,x):  book.\_\_init\_\_(self,x)  def commission(self):  self.comm=self.price\*5/100  print ("Commission of Stockist is %.2f" %self.comm)  class distributor(book):  def \_\_init\_\_(self,x):  book.\_\_init\_\_(self,x)  def commission(self):  self.comm=self.price\*8/100  print ("Commission of Distributor is %.2f" %self.comm)  class retailer(book):  def \_\_init\_\_(self,x):  book.\_\_init\_\_(self,x)  def commission(self):  self.comm=self.price\*10/100  print ("Commission of Retailer is %.2f" %self.comm)  r = stockist(100)  s = distributor(100)  t = retailer(100)  prncomm = [r,s,t]  for c in prncomm:  c.commission() |

**Lecture 10: File Handling**

3 steps: Opening a file 🡪 Performing action on file 🡪 Closing the file

|  |  |
| --- | --- |
| **Opening a file** | **Performing actions on a file** |
| open(file\_name, mode) | |  |  | | --- | --- | | **Method** | **Purpose** | | close() | Close the file, flush all data | | read([n]) | Reads n number of characters from file. | | readline([n]) | Reads next line from file. If n provided, it will read n number of characters. If complete line is read, \n is included. | | readlines([n]) | Reads next lines from file. | | flush() | Flushes all data from internal buffers to OS file. | | write(str) | Writes given string to file. | | writelines(l) | Writes list of strings to file. | | seek(offset, location) | Sets the location of file handler (FH) at offset. f.seek(0) moves FH to beginning of file. f.seek(10,1) moves FH 10 bytes from its current position. f.seek(10,2)moves FH to 10th byte from enf of file. | | tell() | Returns the position of the FH. | |
| |  |  | | --- | --- | | **Mode** | **Description** | | R | Default. Open file for reading. | | W | Creates a file for writing. Overwrites earlier contents if same name. | | A | Opens file for appending contents. Create a new file if not exist. | | r+ | Opens file for reading and writing. File must already exist. | | w+ | Creates new file for reading, writing. Overwrites the contents if file exist. | | a+ | Opens file for reading and appending contents to end of file. Create a new file if not exist. | |

Displaying Information from a file object

|  |  |
| --- | --- |
| **Method** | **Description** |
| fileno() | Returns internal file descriptor used by OS library when working with this file. |
| isatty() | Returns true if the file is connected to the console or keyboard. |
| closed | True if the file is closed. |
| mode | Mode of file that was used to create the file object through the open() function |
| name | Filename that was passed to open() function when creating the file object. |

* **Serialization (Pickling)** is a process of converting structured data into data stream format.
* Lists, tuples. Functions and classes are preserved using ASCII characters between data value.
* Serialization is done when storing data and deserialization is done when retrieving data.

|  |  |
| --- | --- |
| import pickle  class rect:  def \_\_init\_\_(self, x,y):  self.l = x  self.b = y  def rectarea(self):  return "Area of rectangle is",self.l\*self.b  r=rect(5,8)  f = open('studentinfo.bin', 'wb')  pickle.dump(r, f)  f.close()  del r  f = open('studentinfo.bin','rb')  storedobj = pickle.load(f)  print (storedobj.rectarea()) | * An instance r of class rect is created, and its instance variables, l and b, are initialized to 5 and 8. * A binary file, studentinfo.bin, is opened in write mode, and r is pickled and dumped into it. * The file is then closed. * r is deleted after it is copied into the binary file. * To read r from the file and set it back into useable form, the file is opened in read mode, and the instance is read from the file with pickle.load(), unpickled, and assigned to storedobj. * The area of the rectangle is calculated and displayed by calling rectarea() on the storedobj object. |

* **Exception Handling**. Exception occur when certain situation arise in a program.
* try/except: The code that might raise an error is written in the try block, and all the errors and exceptions are handled through the except clause.
* try/finally: The code written in the finally block always executes whether an exception occurs or not. That is, the code that you want to execute in all situations is written in a finally block.

|  |  |
| --- | --- |
| **Syntax** |  |
| try:  statement(s)  except SomeException:  code for handling exception  [else:  statement(s)] |
| **Handling EOFError** |
| import sys  try:  n = input(‘Enter your name ‘)  except EOFError:  print (‘EOF error has occurred’ )  sys.exit(1)  except:  print (‘Some error has occurred’ )  else:  print(‘The name entered is’, n) |

* **Raising an exception**. Exceptions are automatically raised when some undesired situation occurred.
* **Syntax**:

raise customException, statement for customException

|  |
| --- |
| class myException(Exception):  def \_\_init\_\_(self, quantity):  Exception.\_\_init\_\_(self)  self.quantity = quantity  try:  s = int(input('Enter quantity '))  if s <=0 :  raise myException(s)  except EOFError:  print ('You pressed EOF ')  except myException as ex:  print ('myException: The quantity entered is %d,  it must be some positive value' %ex.quantity)  else:  print ('No exception raised.') |

* assert statement is used to place an error-checking statement in program.
* It is convenient to debug a program.
* Return true if all values of variables are as expected.
* AssertionError exception raised when returns false.

|  |  |  |
| --- | --- | --- |
| **File attribute** | **Error message if file not exist.** | **Append content to a file.** |
| f = open("aboutbook.txt", "r")  print ("Name of file:", f.name)  print ("Closed?", f.closed)  print ("Opening mode:", f.mode)  print ("File number descriptor  is:", f.fileno())  f.close()  Output:  Name of file: aboutbook.txt  Closed? False  Opening mode: r  File number descriptor is: 3 | import sys  try:  f = open('about.txt', 'r')  lines = f.read()  except IOError:  print ('File not exist')  sys.exit(1)  f.close()  print (lines) | import sys  matter2 = ''' Its very hot today  Lets have a Cold drink '''  f = open('aboutbook.txt', 'a' )  f.write("\n%s" %matter2)  f.close()  f = open('aboutbook.txt', 'r' )  lines = f.readlines()  f.close()  print('The contents in file:')  for line in lines:  sys.stdout.write(line) |
| **Copying a file** | **Read content of file randomly** | **Delete content from a file** |
| f = open('about.txt', 'r')  lines = f.read()  f.close()  g = open('copyabout.txt', 'w' )  g.write(lines)  g.close()  print('The copy is made')  g = open('copyabout.txt', 'r' )  lines = g.read()  print (lines)  g.close() | f = open('about.txt', 'r')  line=f.readline()  print('A line:', line)  f.seek(5)  line=f.readline()  print('The line from char 6\ till end of line is:', line)  print ('The pointer is at\ location', f.tell())  f.seek(10)  line=f.read(15)  print('The 15 char starting\ at location 11 are as:',\ line) | import sys  f = open('aboutbook.txt', 'r' )  lines = f.readlines()  print('Original content:')  for line in lines:  sys.stdout.write(line)  f.close()  del lines[1:3]  f = open('aboutbook.txt', 'w' )  f.writelines(lines)  f.close()  print('After deleting second and  third line:')  f = open('aboutbook.txt', 'r' )  lines = f.read()  print (lines)  f.close() |
| **Update content of a file** | **Access specific content in file** | **Creating binary file** |
| import sys  f = open('about.txt', 'r')  lines = f.readlines()  print('Original content:')  for line in lines:  sys.stdout.write(line)  f.close()  n=int(input("Line to change: "))  if n <=len(lines):  r=input("Enter new content: ")  lines[n-1]=r+"\n"  f = open('about.txt', 'w' )  f.writelines(lines)  f.close()  print('After updating: ' , n)  f = open('about.txt', 'r' )  lines = f.read()  print (lines)  f.close()  else:  print ("The line number", n, "is not found in the file" ) | import linecache  line=linecache.\ getline('about.txt', 3)  print ('The content of the\ third line is:', line | str = 'Hello World!'  f = open("filebinary.bin","wb" )  f.write(str.encode('utf-8'))  f.close()  f = open("filebinary.bin","rb" )  fcontent=f.read()  f.close()  print('The content in the file is:')  print (fcontent.decode('utf-8')) |
| **TypeError and ZeroDivisionError** | **try/finally block** | **Assertion** |
| from \_\_future\_\_ import division  import sys  n = input('Enter a number ')  if n.isdigit():  n=int(n)  try:  m=15/n  except TypeError as ex:  print('Not entered numeric\  value:', ex)  sys.exit(1)  except ZeroDivisionError as ex:  print('Entered 0 value:', ex)  sys.exit(1)  print ('The result is', m) | import sys  try:  f = open('about.txt', 'r')  try:  lines = f.read()  finally:  f.close()  except IOError:  print ('File about.txt\ does not exist')  sys.exit(1)  print (lines) | n=int(input('Positive value:'))  assert(n >=0),"Entered value is\ not a positive value"  Output:  Positive value: -5  Traceback(most recent call last):  File "D:\python\assertex.py", line 2, in >module>  assert(n >=0), "Entered value is not a positive value"  AssertionError: Entered value is not a positive value  Enter a positive value: 5 |

**Lecture 11: GUI Application Development with PyQt**

* Qt toolkit (Qt) is a cross-platform application and UI framework originally.
* Qt is developed by Trolltech
* Runs on several platforms.
* Also known as widget toolkit because it provides buttons, labels, etc.
* PyQt is a set of Python bindings for Qt toolkit.
* Event-handling mechanism is also known as signals and slots.
* Every widget emits signals when its state change. To perform a task in response to it, a signal has to connect to a slot.
* Slot refers to the method containing the code that you want to be executed on occurrence of a signal.

Writing GUI Applications using PyQt and Qt Designer.

1. Design UI using QT designer and save into a XML file.
2. Compule the design into a Python code using pyuic. Example: pyuic5 MyGui.ui -o MyGui.py
3. Create a main program that imports the UI class (eg from MyGui import \*), defines auxiliary methods (eg. def func1: ...) and connect the signals from widgets to slot method (eg pushButton1.clicked.connect(self.func1)).

* Qt Designer is a quick and easy way to design GUI. It provides predefined templates:

|  |  |
| --- | --- |
|  |  |
| Dialog with buttons at the bottom | From with OK and Cancel buttons in the right bottom corner. |
| Dialog with buttons on the right | From with OK and Cancel buttons on the right side. |
| Dialog without buttons | Empty form and the superclass is QDialog. |
| Main window | Main application window with menu bar and toolbar (removeable). |
| Widget | Form whose superclass is QWidget |

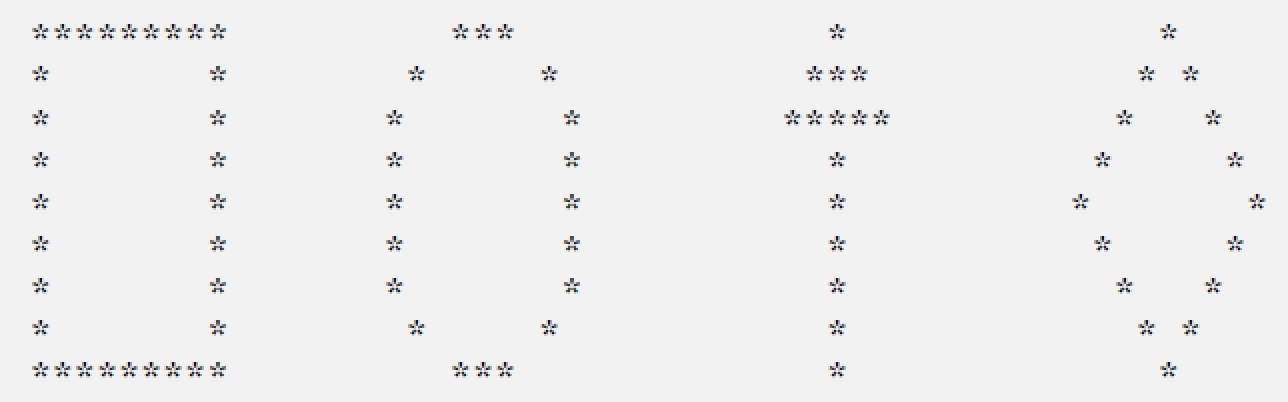
**Tutorial 1**

1. Write a program that allow user to enter the particulars of a student and store them into variables. Then, display the output as follows.

|  |
| --- |
| \*\*\*Welcome to the Department of Electrical Engineering\*\*\*  Student ID: KIE170001  Name: Amin Ali  Date of Birth: 17/01/1997  Height(cm): 174  Weight(kg): 80  \*\*\* End \*\*\* |

|  |
| --- |
| **Sample Answer:** |
| import os #implement clear function  student={}  student["id"]=input("Enter Student ID: ")  student["name"]=input("Enter Name: ")  student["dob"]=input("Enter Date of Birth: ")  student["height"]=input("Enter Height (cm): ")  student["weight"]=input("Enter Weight (kg): ")  clear = lambda: os.system('cls') #declare clear  clear()  print("\*\*\*Welcome to the Department of Electrical Engineering\*\*\*")  print("Student ID: "+student["id"])  print("Name: "+student["name"])  print("Date of Birth: "+student["dob"])  print("Height(cm): "+student["height"])  print("Weight(kg): "+student["weight"])  print("\*\*\* End \*\*\*") |

1. Write a program that prints a box, an oval, an arrow and a diamond in the same line as below. Try to use single, double and triple quotes in quoting below lines. In this question, which type of quote you prefer to use?



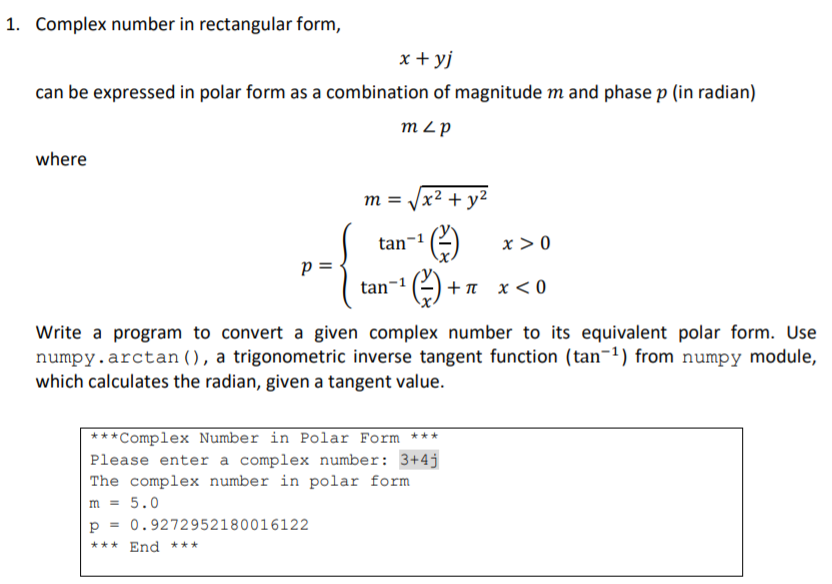
|  |
| --- |
| **Sample Answer:** |
| print('\*\*\*\*\*\*\*\*\*\t\t \*\*\* \t\t \* \t\t \*')  print('\* \*\t\t \* \* \t\t \*\*\* \t\t \* \*')  print('\* \*\t\t\* \*\t\t\*\*\*\*\*\t\t \* \*')  print('\* \*\t\t\* \*\t\t \* \t\t \* \*')  print('\* \*\t\t\* \*\t\t \* \t\t\* \*')  print('\* \*\t\t\* \*\t\t \* \t\t \* \*')  print('\* \*\t\t\* \*\t\t \* \t\t \* \*')  print('\* \*\t\t \* \* \t\t \* \t\t \* \*')  print('\*\*\*\*\*\*\*\*\*\t\t \*\*\* \t\t \* \t\t \*') |

1. Write a program that allow user to calculate area for a chosen shape. An example of execution of such program is as below.

|  |
| --- |
| Area Calculator for Basic Shapes  ================================   1. Square 2. Rectangle 3. Triangle 4. Circle   Choose a shape [1-4] that you want calculate its area : 2  Enter length : 4  Enter base : 2  Area of rectangle is 8.0 |

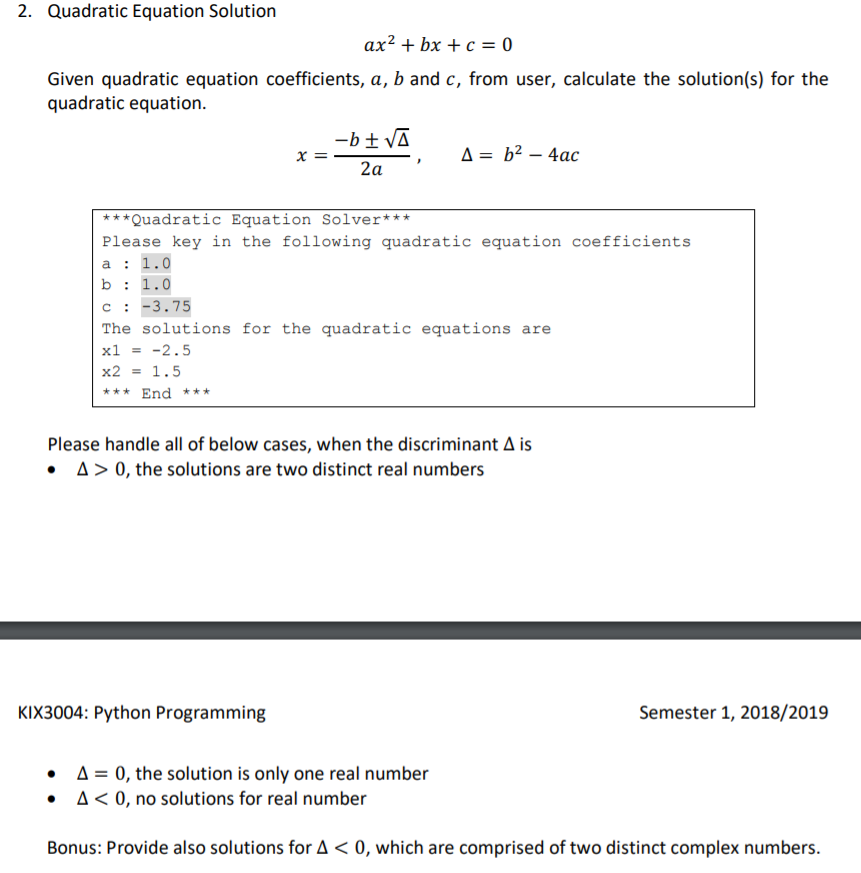
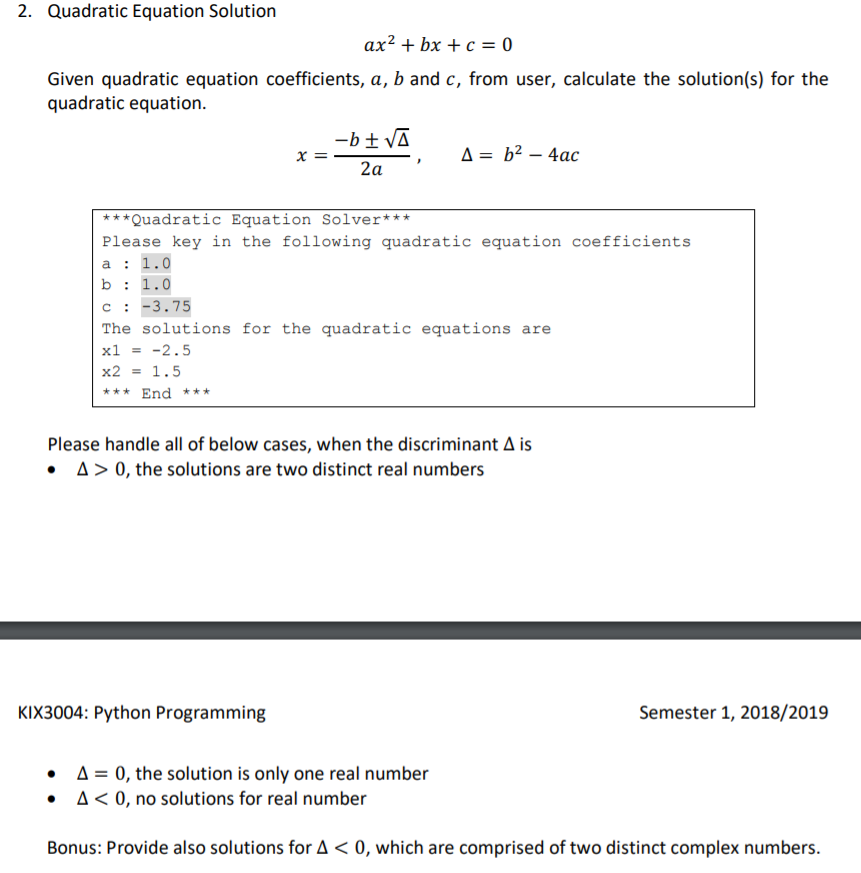
|  |
| --- |
| **Sample Answer:** |
| import math  print('Area Calculator for Basic Shapes')  print('================================',end='\n\n\n\t')  print('1. Square','2. Rectangle','3. Triangle','4. Circle',sep='\n\t')  a=int(input('Choose a shape [1-4] that you want calculate its area : '))  if a==1:  l=float(input('Enter length : '))  b=float(input('Enter base : '))  ans=l\*b  print('Area of square is %f' %ans)  elif a==2:  l=float(input('Enter length : '))  b=float(input('Enter base : '))  ans=l\*b  print('Area of rectangle is %f' %ans)  elif a==3:  l=float(input('Enter length : '))  b=float(input('Enter base : '))  ans=0.5\*l\*b  print('Area of triangle is %f' %ans)  elif a==4:  r=float(input('Enter radius : '))  ans=math.pi\*r\*r  print('Area of circle is %f' %ans)  else:  print('Error!') |

**Tutorial 2**



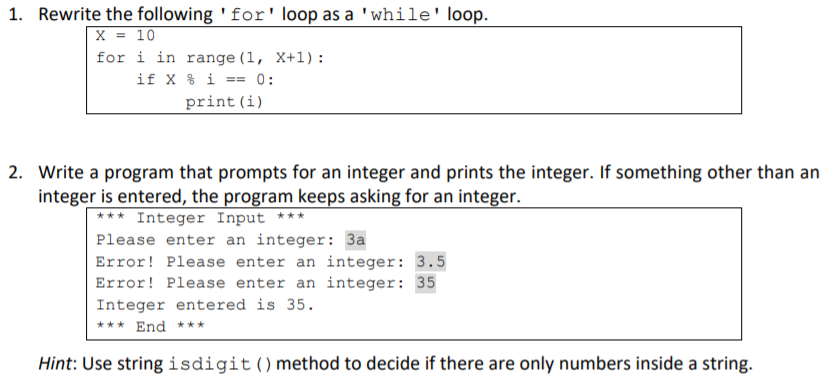
|  |
| --- |
| **Sample Answer (Using math):** |
| import math  print('\*\*\*Complex Number in Polar Form \*\*\*')  a=complex(input('Please enter a complex number: '))  m=math.sqrt(a.real\*\*2+a.imag\*\*2)  if a.real>0:  p=math.atan(a.imag/a.real)  else:  p=math.atan(a.imag/a.real)+math.pi  print('The complex number in polar form ')  print('m = %f\np = %f' %(m,p))  print('\*\*\* End \*\*\*') |

|  |
| --- |
| **Sample Answer (Using numpy):** |
| from numpy import arctan  import math  print('\*\*\*Complex Number in Polar Form \*\*\*')  a=complex(input('Please enter a complex number: '))  m=(a.real\*\*2+a.imag\*\*2)\*\*0.5  if a.real>0:  p=arctan(a.imag/a.real)  else:  p=arctan(a.imag/a.real)+math.pi  print('The complex number in polar form ')  print('m = %f\np = %f' %(m,p))  print('\*\*\* End \*\*\*') |

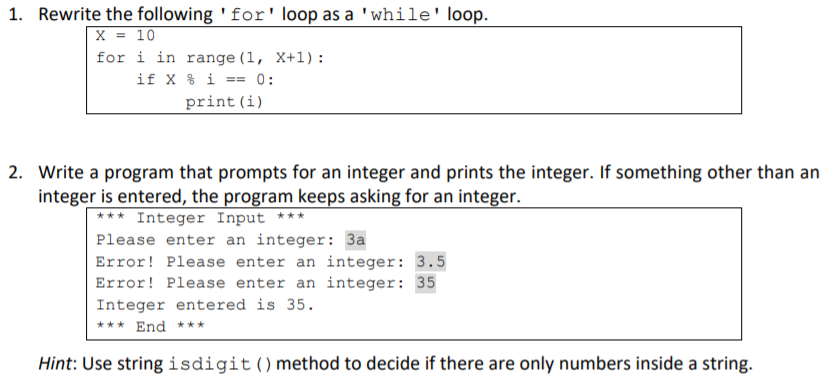


|  |
| --- |
| **Sample Answer:** |
| import math  print('\*\*\*Quadratic Equation Solver\*\*\*')  print('Please key in the following quadratic equation coefficients')  a=float(input('a: '))  b=float(input('b: '))  c=float(input('c: '))  coeff=b\*\*2-4\*a\*c  print('The solutions for the quadratic equations are')  if coeff>0:  x1=(-b+math.sqrt(coeff))/(2\*a)  x2=(-b-math.sqrt(coeff))/(2\*a)  print('x1 = %f\nx2 = %f' %(x1,x2))  elif coeff==0:  x1=(-b+math.sqrt(coeff))/(2\*a)  x2=x1  print('x1 = %f\nx2 = %f' %(x1,x2))  else:  x1r=(-b/(2\*a))  x1i=((math.sqrt(-coeff))/(2\*a))  x2r=(-b/(2\*a))  x2i=((math.sqrt(-coeff))/(2\*a))  print('x1 = %f+%fj' %(x1r,x1i))  print('x2 = %f-%fj' %(x2r,x2i))  print('\*\*\* End \*\*\*') |

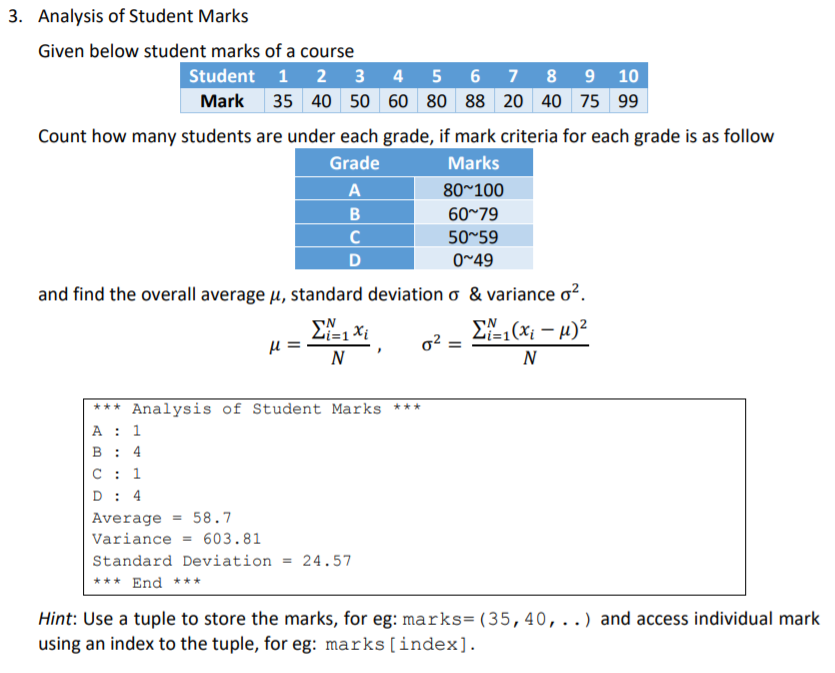
**Tutorial 3**

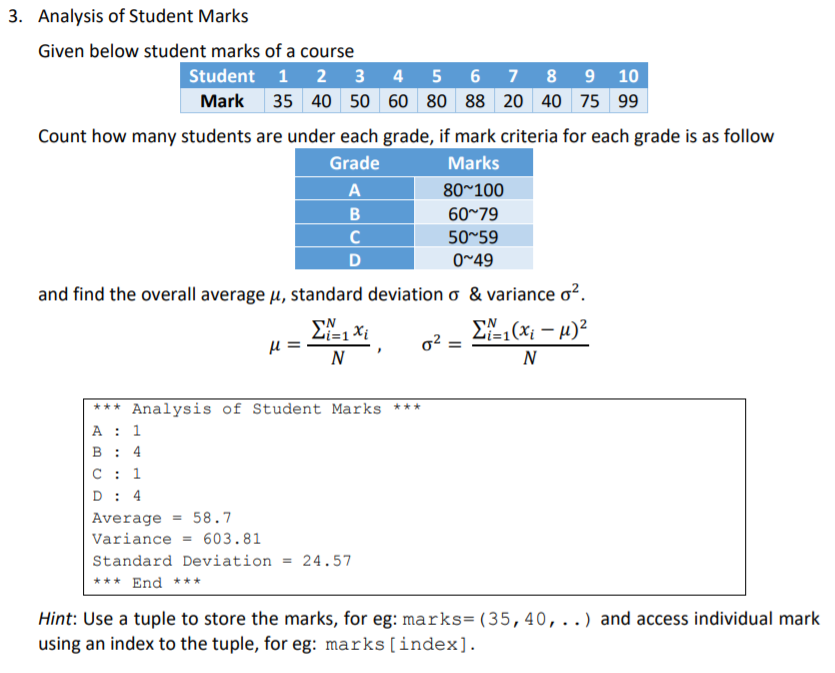


|  |
| --- |
| **Sample Answer:** |
| X=10  i=1  while i<X+1:  if X%i==0:  print(i)  i=i+1 |



|  |
| --- |
| **Sample Answer:** |
| print('\*\*\* Integer Input \*\*\*')  while 1:  a=input('Please enter an integer: ')  if a.isdigit():  break  print('Error!',end=' ')  print('Integer entered is ',a)  print('\*\*\* End \*\*\*') |





|  |
| --- |
| **Sample Answer:** |
| import math  marks=(35,40,50,60,80,88,20,40,75,99)  miu=sum(marks)/len(marks)  a,b,c,d,xmiu=0,0,0,0,0  for i in range(0,len(marks)):  if marks[i]>=80:  a=a+1  elif marks[i]>=60:  b=b+1  elif marks[i]>=50:  c=c+1  else:  d=d+1  xmiu=xmiu+(marks[i]-miu)\*\*2  print('\*\*\* Analysis of Student Marks \*\*\*')  print('A : %i\nB : %i\nC : %i\nD : %i' %(a,b,c,d))  print('Average = %.1f' %miu)  print('Varience = %.2f' %(xmiu/len(marks)))  print('Standard Deviation = %.2f' %(math.sqrt(xmiu/len(marks))))  print('\*\*\* End \*\*\*') |

|  |
| --- |
| **Sample Answer (using numpy):** |
| import numpy as np  marks=np.array([35,40,50,60,80,88,20,40,75,99])  a,b,c,d=0,0,0,0  for i in range(0,len(marks)):  a=a+1 if (marks[i]>=80) else a  b=b+1 if (marks[i]>=60 and marks[i]<=79) else b  c=c+1 if (marks[i]>=50 and marks[i]<=59) else c  d=d+1 if (marks[i]<=49) else d  print('\*\*\* Analysis of Student Marks \*\*\*')  print('A : %i\nB : %i\nC : %i\nD : %i' %(a,b,c,d))  print('Average = %.1f' %np.average(marks))  print('Varience = %.2f' %np.var(marks))  print('Standard Deviation = %.2f' %np.std(marks))  print('\*\*\* End \*\*\*') |

**Tutorial 4**

1. Given x = [1, 2, 3], write the Python code to:
2. Create a list y such that changing x also changing y
3. Create a list y such that changing x does not change y

|  |
| --- |
| x,y: [1, 555, 3] [1, 555, 3]  x,y: [1, 555, 3] [1, 2, 3] |

|  |
| --- |
| **Sample Answer:** |
| #Question 1(a)  x=[1,2,3]  y=x  x[1]=555  print("x,y: ",x,y)  #Question 1(b)  x=[1,2,3]  y=x[:]  x[1]=555  print("x,y: ",x,y) |

1. Come up with four different ways to create list of 25 1’s without simply typing 25 1’s(Hint: You can use for and while loop with append() method for two of the ways)

[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

|  |
| --- |
| **Sample Answer:** |
| list1,list2=[],[]  for i in range (25):  list1.append(1)  while len(list2)<25:  list2.append(1)  list3=[1]\*25  list4=[1 for i in range(25)]  print(list1,list2,list3,list4,sep="\n") |

1. Write a code using a for loop that take a string S as an argument and returns S in reversed order. For example if S = “Python”, it should return “nohtyP”

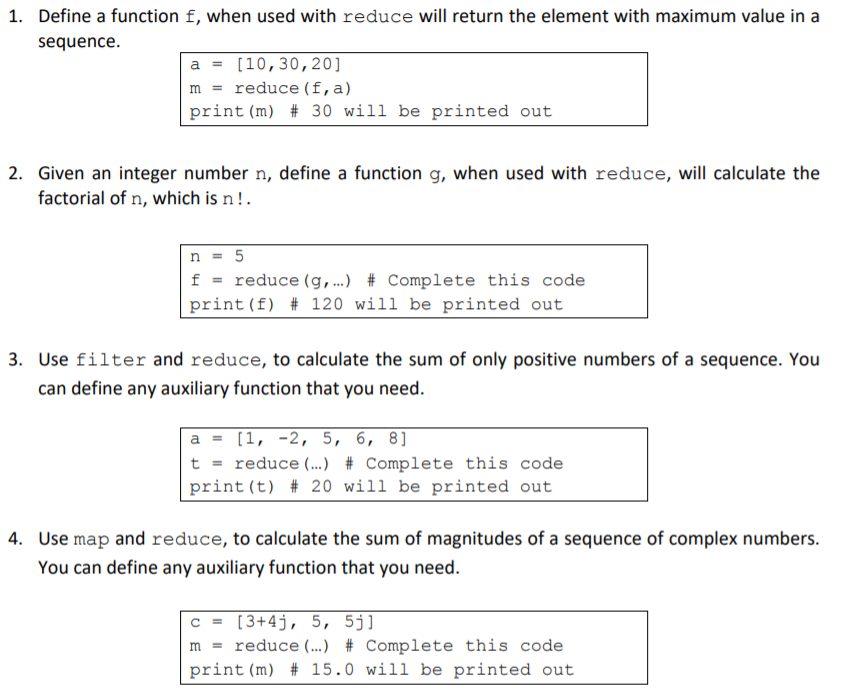
Enter a string : Python

The reversed of Python is nohtyP

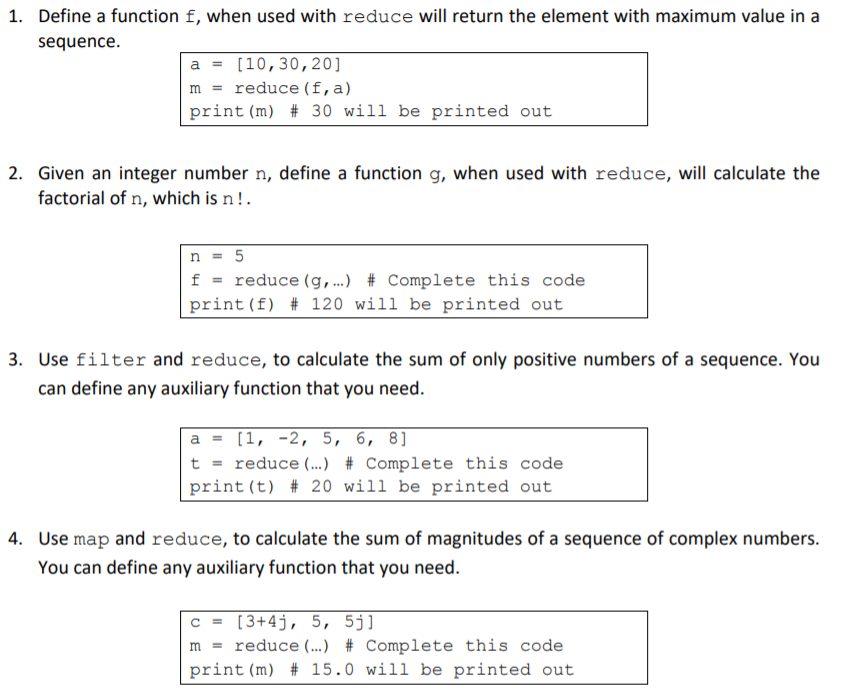
|  |
| --- |
| **Sample Answer 1:** |
| S=input("Enter a string : ")  s=""  for i in range(1,len(S)+1):  s+=S[len(S)-i]  print("The reversed of ",S," is ",s) |

|  |
| --- |
| **Sample Answer 2:** |
| S=input("Enter a string : ")  print("The reversed of ",S," is ","".join(reversed(S))) |

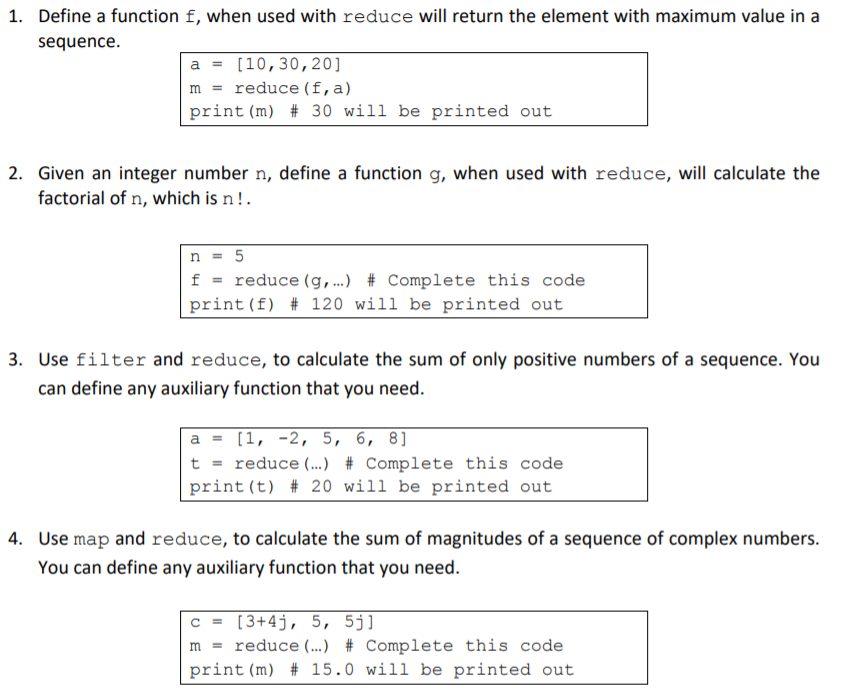
**Tutorial 5**



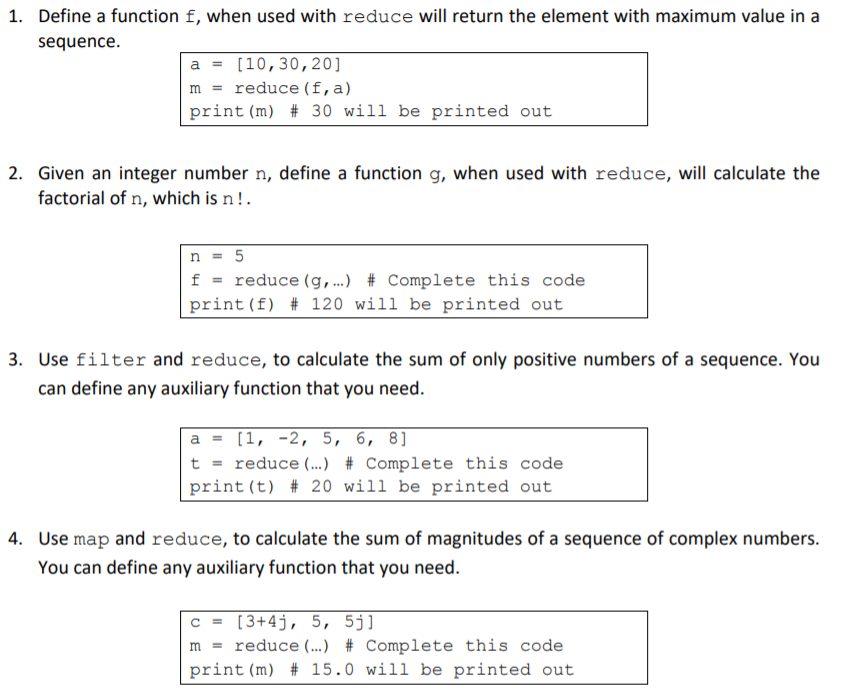
|  |
| --- |
| **Sample Answer:** |
| from functools import reduce  a=[10,30,20]  m=reduce(lambda x,y: x if x>y else y, a )  print(m) |



|  |
| --- |
| **Sample Answer:** |
| from functools import reduce  n=5  f=reduce(lambda x,y:x\*y, range(1,n+1) )  print(f) |

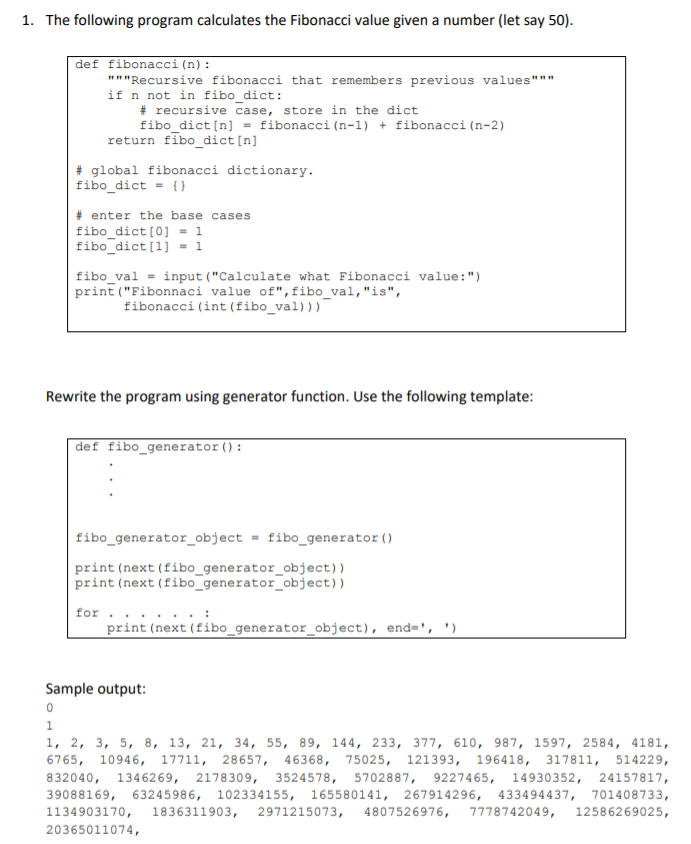


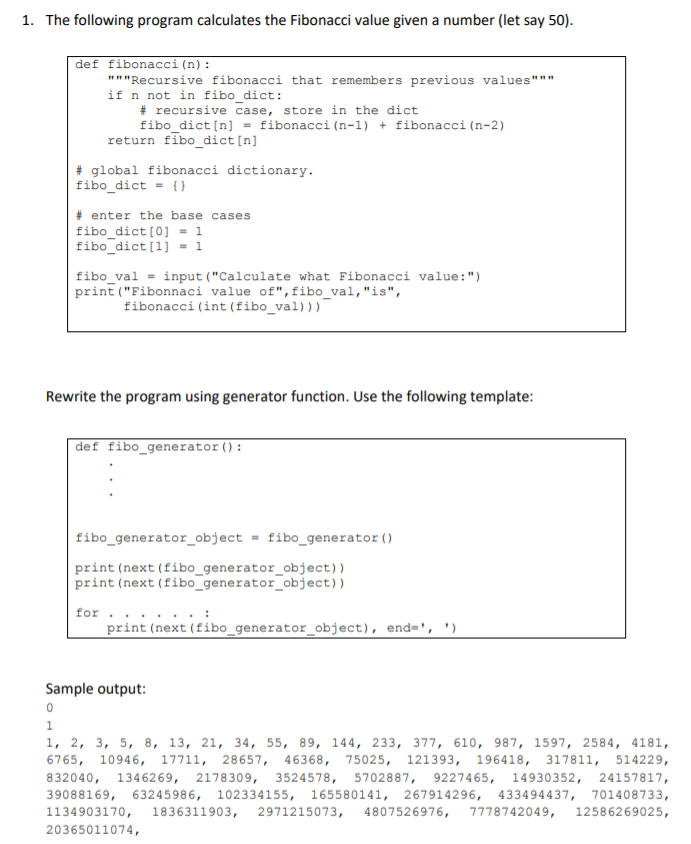
|  |
| --- |
| **Sample Answer:** |
| from functools import reduce  a=[1,-2,5,6,8]  t=reduce(lambda x,y: x+y, filter(lambda a:a>0,a))  print(t) |

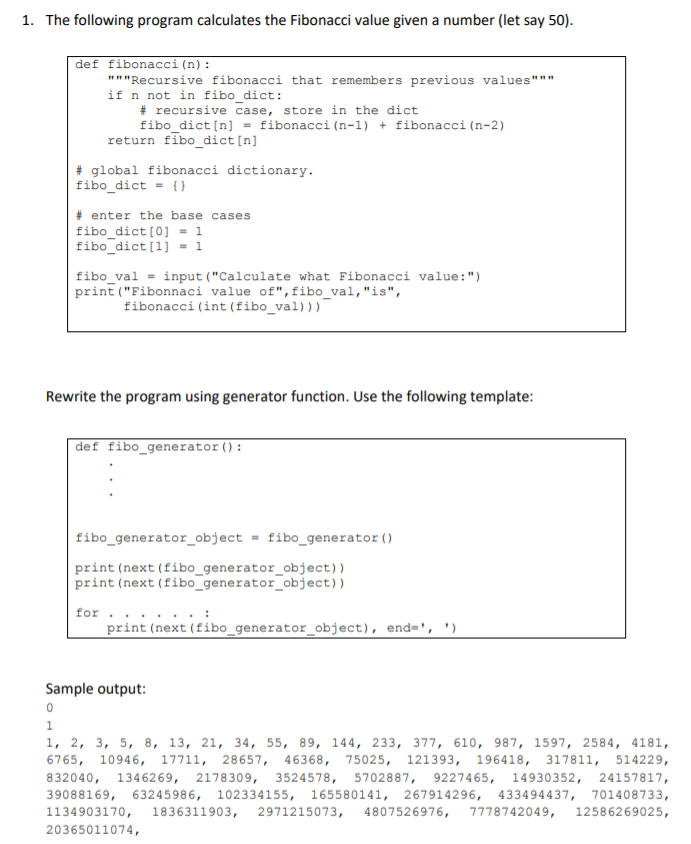


|  |
| --- |
| **Sample Answer:** |
| from functools import reduce  c=[3+4j,5,5j]  m=reduce(lambda a,b:a+b,map(lambda a:(a.real\*\*2+a.imag\*\*2)\*\*0.5,c))  print(m) |

**Tutorial 6**

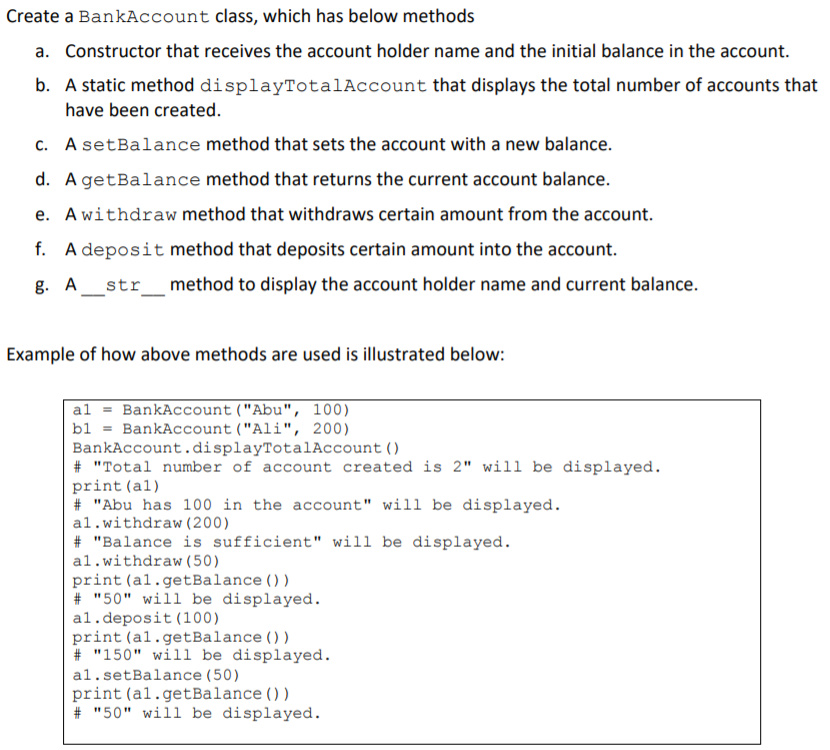


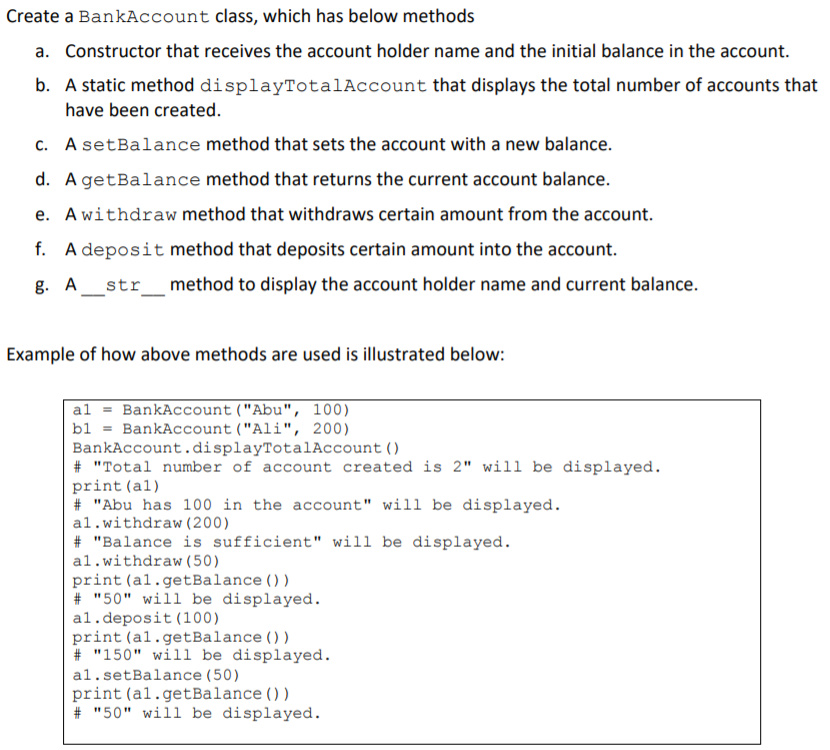


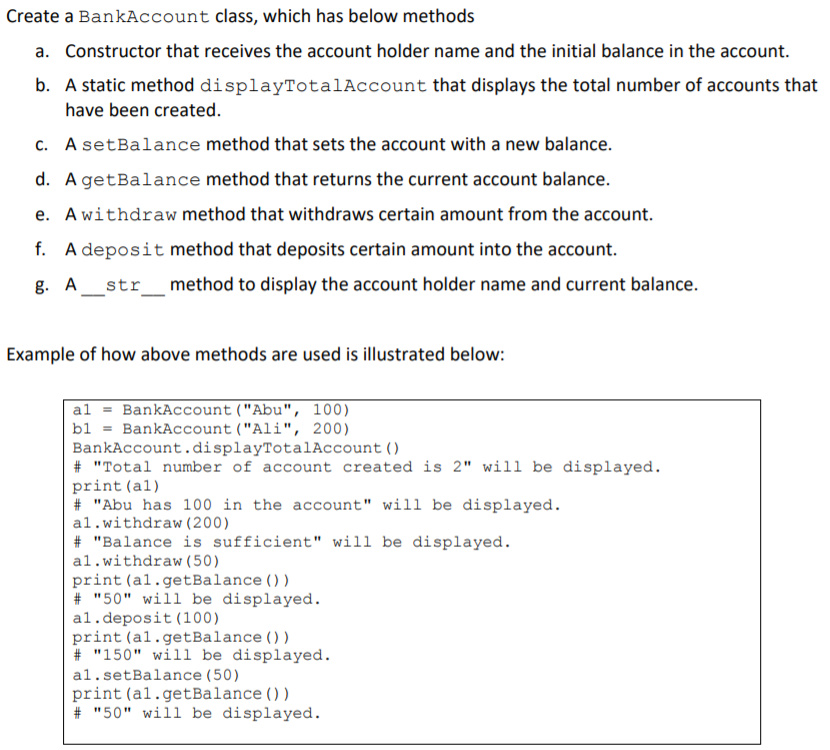


|  |
| --- |
| **Sample Answer:** |
| def fibo\_generator():  a,b=0,1  while 1:  yield a  a,b=b,a+b  fibo\_generator\_object = fibo\_generator()  print(next(fibo\_generator\_object))  print(next(fibo\_generator\_object))  for \_ in range (50) :  print(next(fibo\_generator\_object), end=', ') |

**Tutorial 7**

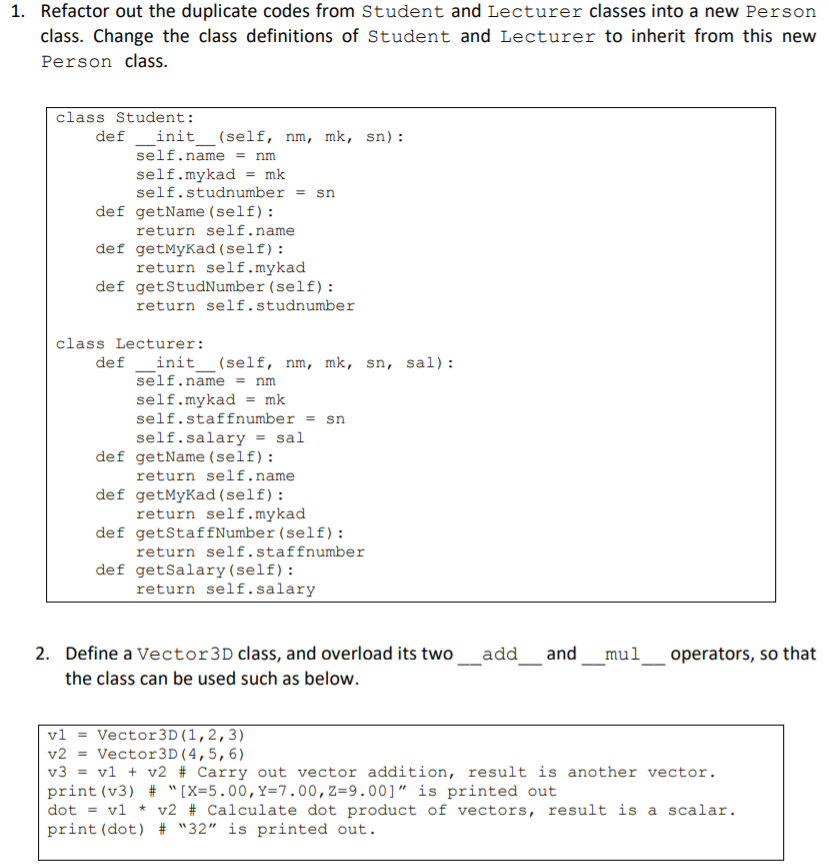




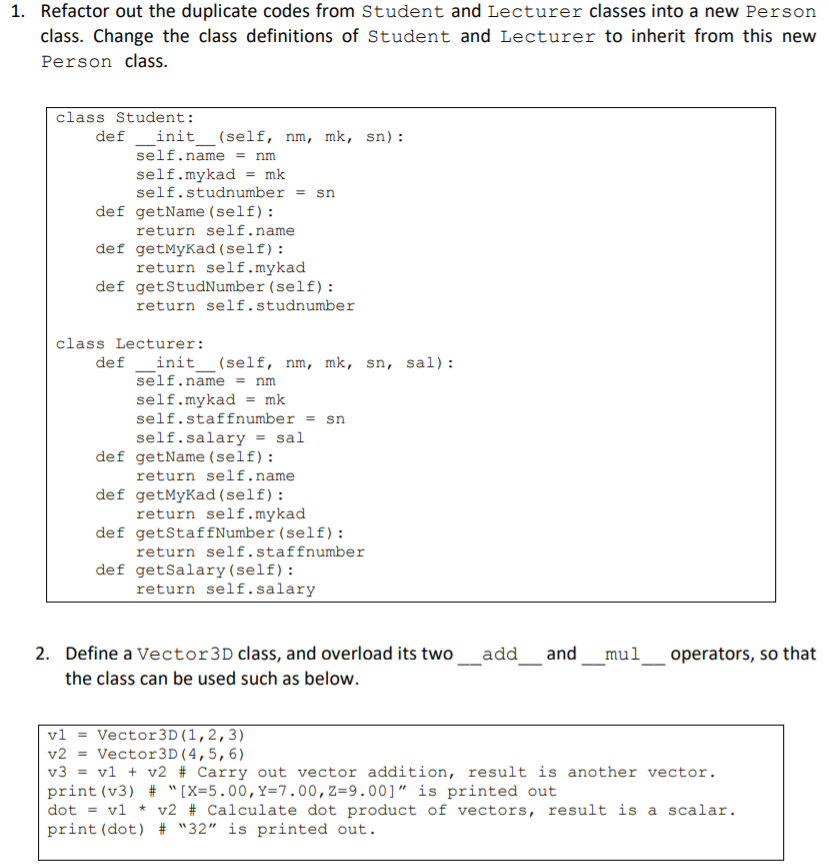


|  |
| --- |
| **Sample Answer:** |
| class BankAccount:  count=0  def \_\_init\_\_(self,a,b):  self.name=a  self.bal=b  BankAccount.count += 1  def \_\_str\_\_(self):  return '%s has %i in the account' %(self.name,self.bal)  @staticmethod  def displayTotalAccount():  print('Total number of account created is %i' %(BankAccount.count))  def setBalance(self,i):  self.bal=i  def getBalance(self):  return self.bal  def withdraw(self,i):  if (i>self.bal):  print('Balance is insufficient')  else:  self.bal-=i  def deposit(self,i):  self.bal+=i    a1 = BankAccount("Abu", 100)  b1 = BankAccount("Ali", 200)  BankAccount.displayTotalAccount()  print(a1)  a1.withdraw(200)  a1.withdraw(50)  print(a1.getBalance())  a1.deposit(100)  print(a1.getBalance())  a1.setBalance(50)  print(a1.getBalance()) |

**Tutorial 8**

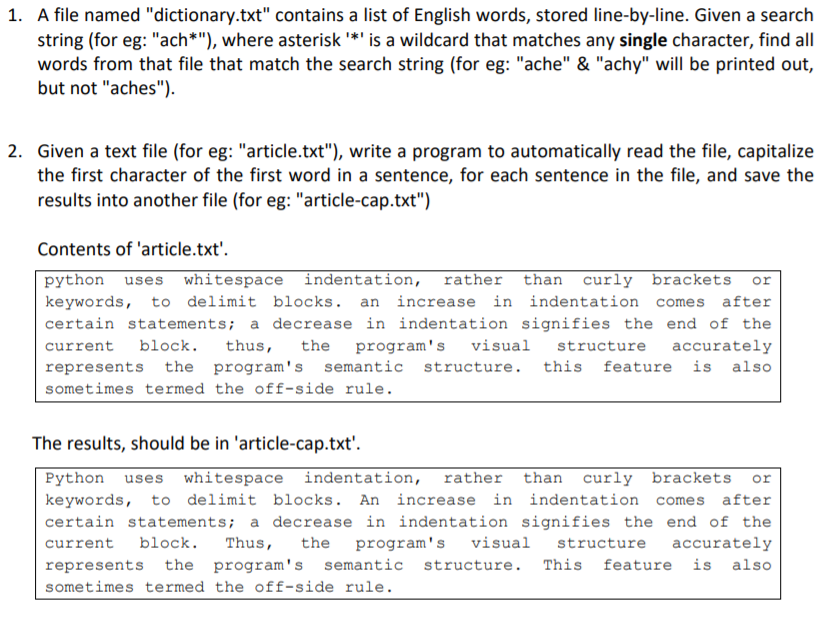


|  |
| --- |
| **Sample Answer:** |
| class Person:  def \_\_init\_\_(self,nm,mk,sn):  self.name=nm  self.mykad=mk  self.id=sn  def getName(self):  return self.name  def getMyKad(self):  return self.mykad  class Student(Person):  def \_\_init\_\_(self,nm,mk,sn):  Person.\_\_init\_\_(self,nm,mk,sn)  def getStudNumber(self):  return self.id  class Lecturer(Person):  def \_\_init\_\_(self,nm,mk,sn,sal):  Person.\_\_init\_\_(self,nm,mk,sn)  self.salary=sal  def getStaffNumber(self):  return self.id  def getSalary(self):  return self.salary |

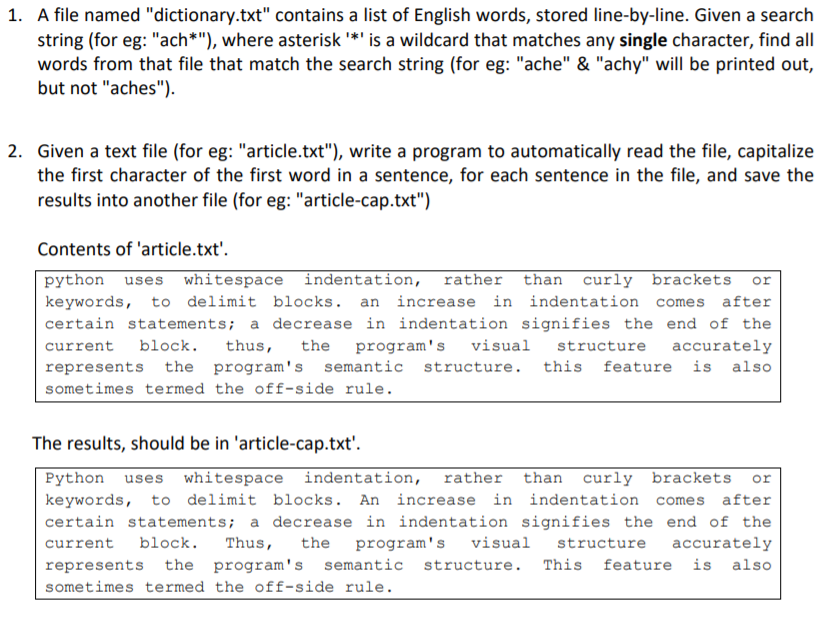


|  |
| --- |
| **Sample Answer:** |
| class Vector3D:  def \_\_init\_\_(self,a,b,c):  self.x=a  self.y=b  self.z=c  def \_\_add\_\_(self,other):  return Vector3D(self.x+other.x,self.y+other.y,self.z+other.z)  def \_\_mul\_\_(self,other):  return (self.x\*other.x)+(self.y\*other.y)+(self.z\*other.z)  def \_\_str\_\_(self):  return "[X=%.2f,Y=%.2f,Z=%.2f]" %(self.x,self.y,self.z)  v1=Vector3D(1,2,3)  v2=Vector3D(4,5,6)  v3=v1+v2  print(v3)  dot=v1\*v2  print(dot) |

**Tutorial 9**



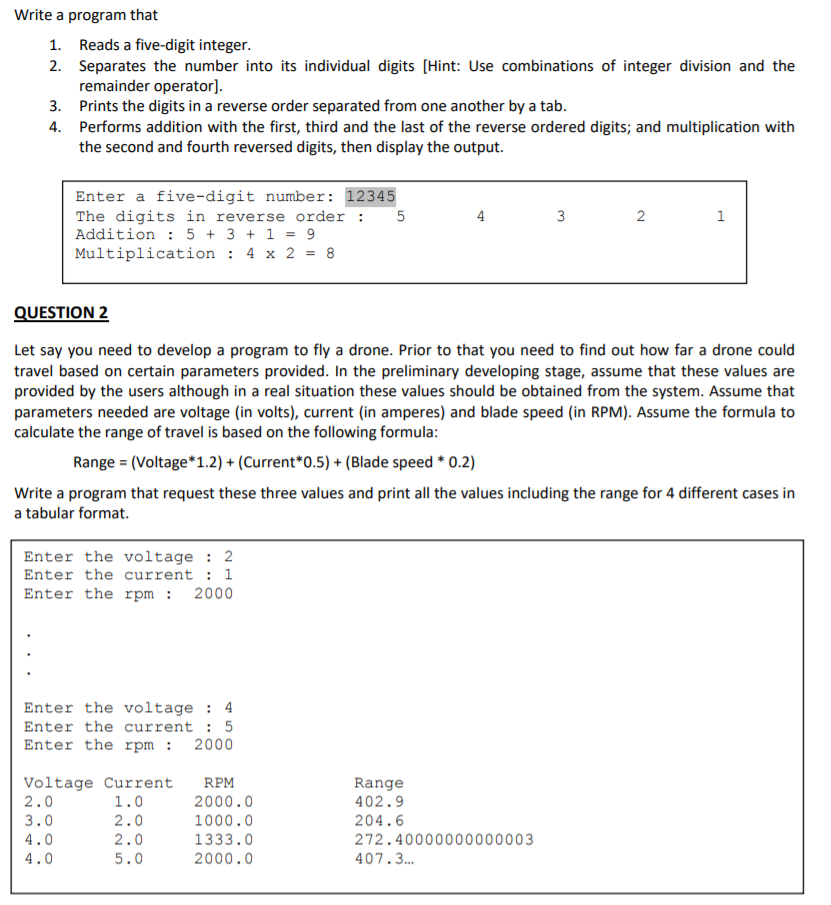
|  |
| --- |
| **Sample Answer:** |
| f=open('dictionary.txt')  lines=f.readlines()  f.close()  for l in lines:  if(len(l)==5 and l[0]=='a' and l[1]=='c' and l[2]=='h'):  print(str(l)) |



|  |
| --- |
| **Sample Answer:** |
| f=open('article.txt','r')  l=f.read()  re=""  l=l.split('. ')  for i in l:  re+=i[0].upper()+i[1:]+'. '  g=open('article-cap.txt','w')  g.write(re[:-3]) |

**Test 1**

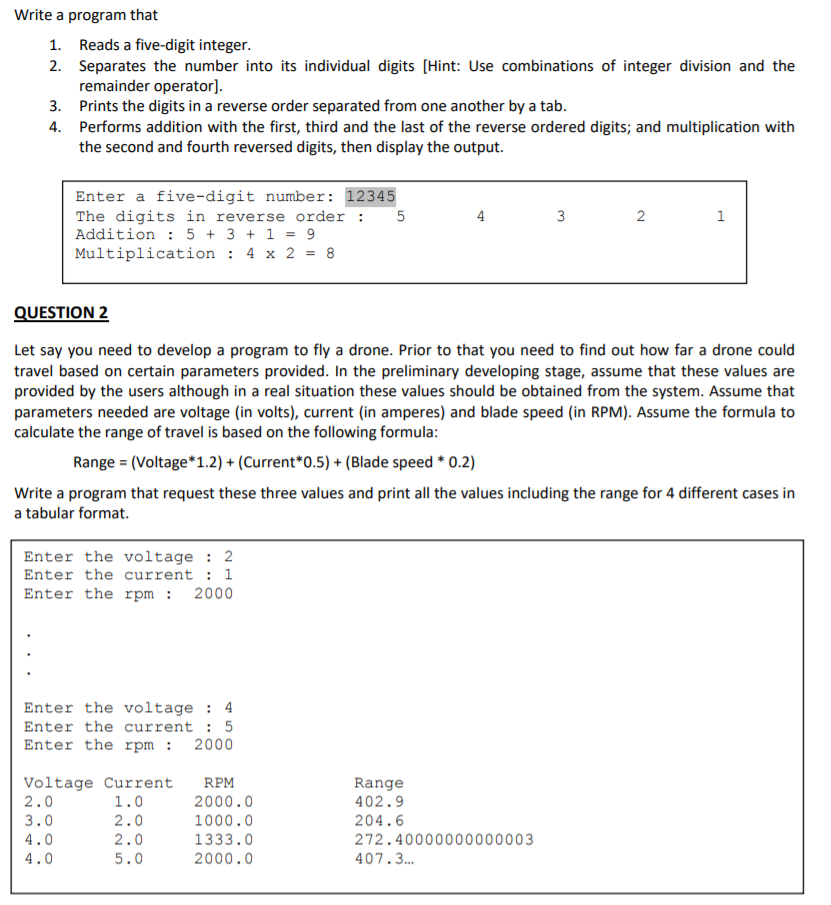
Question 1



|  |
| --- |
| **Sample Answer 1:** |
| a=int(input('Enter a five-digit number: '))  num=[]  for i in range(5):  num.append(int(a/(10\*\*i)%10))  print('The digits in reverse order : \t',end="")  for i in range(5):  print(num[i],end="\t")  print('',end='\n')  print('Addition : ',num[0],'+',num[2],'+',num[4],'=',num[0]+num[2]+num[4])  print('Multiplication :',num[1],'x',num[3],'=',num[1]\*num[3]) |

|  |
| --- |
| **Sample Answer 2:** |
| a=str(input('Enter a five-digit number: '))  print('The digits in reverse order :',end="")  for i in range(len(a)):  print((a)[::-1][i],end="\t")  print('\nAddition : ',(a)[::-1][0],'+',(a)[::-1][2],'+',(a)[::-1][4],'=',\  int((a)[::-1][0])+int((a)[::-1][2])+int((a)[::-1][4]))  print('Multiplication : ',(a)[::-1][1],'x',(a)[::-1][3],'=',\  int((a)[::-1][1])\*int((a)[::-1][3])) |

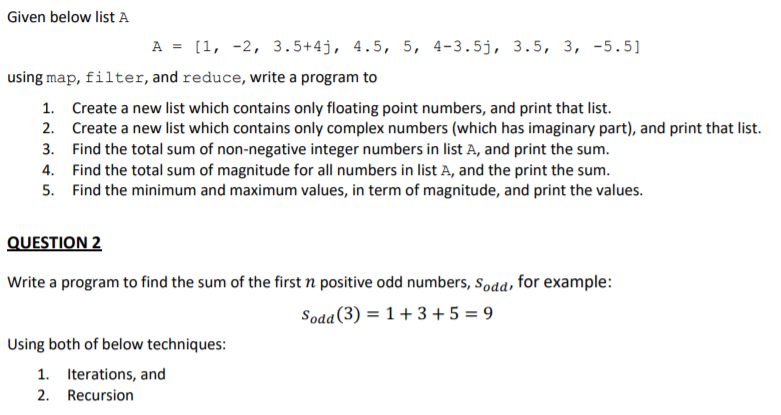
Question 2



|  |
| --- |
| **Sample Answer:** |
| v,c,r=[],[],[]  for \_ in range(4):  v.append(float(input('Enter the voltage : ')))  c.append(float(input('Enter the current : ')))  r.append(float(input('Enter the rpm : ')))    print('Voltage\tCurrent\t RPM\t\t Range')  for i in range(4):  q=float((v[i]\*1.2)+(c[i]\*0.5)+(r[i]\*0.2))  print(v[i],c[i],r[i],sep="\t ",end="")  print('\t\t',q) |

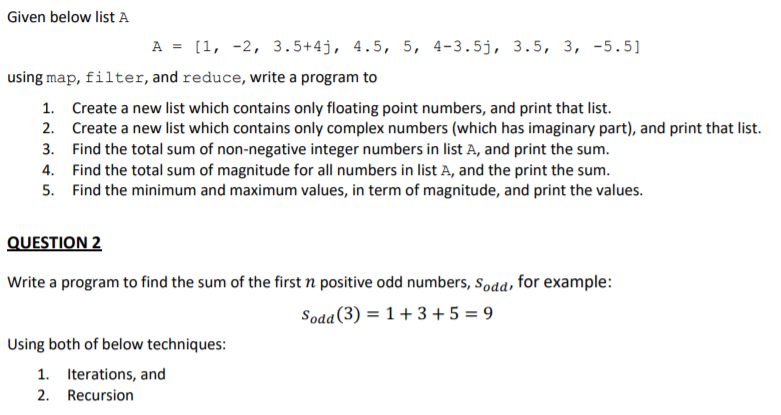
**Test 2**

Question 1



|  |
| --- |
| **Sample Answer:** |
| from functools import reduce  A=[1,-2,3.5+4j,4.5,5,4-3.5j,3.5,3,-5.5]  #Question 1  floating=list(filter(lambda x:x if x.imag==0 and type(x)==float else 0,A))  print(floating)  #Question 2  comp=list(filter(lambda x:x if x.imag!=0 else 0,A))  print(comp)  #Question 3  sum\_non=reduce(lambda x,y:x+y,filter(lambda x: x if type(x)==int and x>=0 else 0,A))  print(sum\_non)  #Question 4  sum\_mag=reduce(lambda x,y:x+y,map(lambda a:(a.real\*\*2+a.imag\*\*2)\*\*0.5 if type(a)==complex else a,A))  print(sum\_mag)  #Question 5  maxi=reduce(lambda x,y: x if x>y else y, map(lambda a:(a.real\*\*2+a.imag\*\*2)\*\*0.5,A))  print(maxi)  mini=reduce(lambda x,y: y if x>y else x, map(lambda a:(a.real\*\*2+a.imag\*\*2)\*\*0.5,A))  print(mini) |

Question 2



|  |
| --- |
| **Sample Answer:** |
| def sum\_iter(n):  t=0  i=1  while i<(2\*n)+1:  if i%2==1:  t+=i  i+=2  return t  def sum\_rec(n):  if n==1:  return 1  else:  return 2\*n-1+sum\_rec(n-1)    print(sum\_iter(3))  print(sum\_rec(3)) |