10.009 The Digital World

Term 3. 2016

Problem Set 3 (for Week 3)

Last update: January 29, 2016

Due dates:

- **Problems: Cohort sessions**: Following week: Monday 11:59pm.
- **Problems: Homework**: Same as for the cohort session problems.
- **Problems: Exercises**: These are practice problems and will not be graded. You are encouraged to solve these to enhance your programming skills. Being able to solve these problems will likely help you prepare for the midterm examination.

Objectives:

1. Learn to use if-elif-else statement.
2. Learn the definitions and literals for list.
3. Learn the `while` statements.
4. Learn basic techniques for program debugging.
5. Learn to write simple loops.

Note: Solve the programming problems listed below using the IDLE or Canopy editor. Make sure you save your programs in files with suitably chosen names and in an newly created directory. In each problem find out a way to test the correctness of your program. After writing each program, test it, debug it if the program is incorrect, correct it, and repeat this process until you have a fully working program. Show your working program to one of the cohort instructors.

Problems: Cohort sessions

1. *Program tracing:* What is printed when the following Python code snippets are executed? For each problem assume three cases: Case 1: $x < y$, Case 2: $x > y$, and Case 3: $x = y$.

   (a)
   ```
   x=int(raw_input("Enter:"))
   y=int(raw_input("Enter:"))
   if x<y:
       print x
   elif x>y:
       print y
   else:
       print x+y
   ```

   (b)
   ```
   x=int(raw_input("Enter:"))
   y=int(raw_input("Enter:"))
   if x<y:
       print x+y
   elif x>y:
       print y
       if(x+y)>50:
           print x-y
   else:
       print x
   ```

2. *Functions and conditions:* Write a function named `letterGrade()` that takes an integer argument, which represents a *mark* of a student. Return 'A' if $mark \geq 90$, 'B' if $80 \leq mark < 90$, 'C' if $70 \leq mark < 80$, 'D' if $60 \leq mark < 70$, and 'E' if $mark < 60$. Else, you return `None` if it is not a valid mark.

   ```
   >>>print letterGrade(102)
   None

   >>>print letterGrade(100)
   A

   >>>print letterGrade(83)
   B

   >>>print letterGrade(75)
   C

   >>>print letterGrade(67)
   D

   >>>print letterGrade(52)
   E

   >>>print letterGrade(-2)
   None
   ```

3. *Functions: largest number:* Write a function named `check1()` that takes two numbers, say $n1$ and $n2$, as inputs. If $n1 > n2$ the function returns `True` otherwise it returns `False`.

```
>>>print check1(2,-1)
True

>>>print check1(-1,2)
False

>>>print check1(2,2)
False
```

4. *Loops, summation*: Write a function named `listSum()` that takes a list of values as input and returns the sum of all the values. For example, if the input is $[3, 5.0, 9]$ the function must return 17.0. Return 0.0 if the list is empty.

```
print ("Test case 1: [4.25,5.0,10.75]")
ans=listSum([4.25,5.0,10.75])
print ans

print ("Test case 2: [5.0]")
ans=listSum([5.0])
print ans

print ("Test case 3: []")
ans=listSum([])
print ans
```

The output should be:

```
Test case 1: [4.25,5.0,10.75]
20.0
Test case 2: [5.0]
5.0
Test case 3: []
0.0
```

5. *Loops: maximum:* Write a function named `maxList()` that takes list of integers as input and returns the minimum and maximum values in the list. Return `(None, None)` if the list is empty. Note that the maximum and minimum integers in Python are given by, respectively, the constants `sys.maxint` and `-sys.maxint-1`. You must use loops rather than any built-in function.

```
print ("Test case 1: [1,2,3,4,5]")
ans=maxList([1,2,3,4,5])
print ans

print ("Test case 2: [1,1,3,0]")
ans=maxList([1,1,3,0])
print ans

print ("Test case 3: [3,2,1]")
ans=maxList([3,2,1])
print ans

print ("Test case 4: [0,10]")
ans=maxList([0,10])
print ans
```

```
print ("Test case 5: [1]")
ans=maxList([1])
print ans

print ("Test case 6: []")
ans=maxList([])
print ans

print ("Test case 7: [1,1,1,1,1]")
ans=maxList([1,1,1,1,1])
print ans
```

The output should be:

```
Test case 1: [1,2,3,4,5]
(1, 5)
Test case 2: [1,1,3,0]
(0, 3)
Test case 3: [3,2,1]
(1, 3)
Test case 4: [0,10]
(0, 10)
Test case 5: [1]
(1, 1)
Test case 6: []
(None, None)
Test case 7: [1,1,1,1,1]
(1, 1)
```

6. *Functions: palindrome:* A number is a *palindrome number* if it reads the same from left to right as from right to left. For example, 1, 22, 12321, 441232144 are palindrome numbers. Write a function that takes an input integer and returns a boolean value that indicates whether the integer is a palindrome number.

```
print ("Test case 1: 1")
ans=isPalindrome(1)
print ans

print ("Test case 2: 22")
ans=isPalindrome(22)
print ans

print ("Test case 3: 12321")
ans=isPalindrome(12321)
print ans

print ("Test case 4: 441232144")
ans=isPalindrome(441232144)
print ans

print ("Test case 5: 441231144")
ans=isPalindrome(441231144)
print ans

print ("Test case 6: 144")
ans=isPalindrome(144)
print ans
```

```
print ("Test case 7: 12")
ans=isPalindrome(12)
print ans
```

The output should be:

```
Test case 1: 1
True
Test case 2: 22
True
Test case 3: 12321
True
Test case 4: 441232144
True
Test case 5: 441231144
False
Test case 6: 144
False
Test case 7: 12
False
```

Problems: Homework

1. *Functions: Arguments:* Write a function named `check2()` that takes four integers $n1$, $n2$, $n3$, and x as inputs. The function returns `True` if x is greater than $n1$ and $n2$ but is less than $n3$, otherwise it returns `False`.

```
print '''Test case 1: check2(1,4,8,7)'''
print '''ans = True'''
ans=check2(1,4,8,7)
print ans

print '''Test case 2: check2(10,4,8,7)'''
print '''ans = False'''
ans=check2(10,4,8,7)
print ans

print '''Test case 3: check2(1,10,8,7)'''
print '''ans = False'''
ans=check2(1,10,8,7)
print ans

print '''Test case 4: check2(1,4,5,7)'''
print '''ans = False'''
ans=check2(1,4,5,7)
print ans
```

The output should be:

```
Test case 1: check2(1,4,8,7)
ans = True
True
Test case 2: check2(10,4,8,7)
ans = False
False
Test case 3: check2(1,10,8,7)
ans = False
False
```

```
Test case 4: check2(1,4,5,7)
ans = False
False
```

2. *Functions:* Write a function named `tempConvert()` that takes two arguments. The first argument is a string 'C' or 'F' and the second argument is a number (an integer or a float). If the first argument is a 'C' then return the centigrade equivalent of the input number. If the first argument is an 'F' then return the fahrenheit equivalent of the input number. If the input string is neither 'C' nor 'F' then return `None`. To do the requested conversion, use the `fToC()` and `cToF()` functions that have been written for you.

```
print '''Test case 1: F = 32'''
ans=tempConvert('F', 32)
print ans

print '''Test case 2: F = -40'''
ans=tempConvert('F', -40)
print ans

print '''Test case 3: F= 212'''
ans=tempConvert('F', 212)
print ans

print '''Test case 4: C = 0'''
ans=tempConvert('C', 0)
print ans

print '''Test case 5: C = -40'''
ans=tempConvert('C', -40)
print ans

print '''Test case 6: C = 100'''
ans=tempConvert('C', 100)
print ans

print '''Test case 7: Neither 'C' nor 'F'''
ans=tempConvert('', 0)
print ans

print '''Test case 8: Neither 'C' nor 'F'''
ans=tempConvert('A', 0)
print ans
```

The output should be:

```
Test case 1: F = 32
89.6
Test case 2: F = -40
-40.0
Test case 3: F= 212
413.6
Test case 4: C = 0
-17.7777777778
Test case 5: C = -40
-40.0
Test case 6: C = 100
37.7777777778
```

```
Test case 7: Neither 'C' nor 'F
None
Test case 8: Neither 'C' nor 'F
None
```

3. *Functions: even:* Write a function `even` that takes a list and return a new list containing all even numbers in the original list.

```
print 'getEvenNumber([1,2,3,4,5])'
ans=getEvenNumber([1,2,3,4,5])
print ans

print 'getEvenNumber([11,22,33,44,55])'
ans=getEvenNumber([11,22,33,44,55])
print ans

print 'getEvenNumber([10,20,30,40,50])'
ans=getEvenNumber([10,20,30,40,50])
print ans

print 'getEvenNumber([11,21,31,41,51])'
ans=getEvenNumber([11,21,31,41,51])
print ans
```

The output should be:

```
getEvenNumber([1,2,3,4,5])
[2, 4]
getEvenNumber([11,22,33,44,55])
[22, 44]
getEvenNumber([10,20,30,40,50])
[10, 20, 30, 40, 50]
getEvenNumber([11,21,31,41,51])
[]
```

4. *Functions: Prime:* Write a function that takes an integer argument and returns a boolean value that indicates whether the number is a prime number. (Hint: you can try to find whether there exists a number that divides the number under query. If the answer is yes, then it is not a prime number. Otherwise it is a prime number.)

```
print 'isPrime(2)'
ans=isPrime(2)
print ans

print 'isPrime(3)'
ans=isPrime(3)
print ans

print 'isPrime(7)'
ans=isPrime(7)
print ans

print 'isPrime(9)'
ans=isPrime(9)
print ans

print 'isPrime(21)'
```

```
ans=isPrime(21)
print ans
```

The output should be:

```
isPrime(2)
True
isPrime(3)
True
isPrime(7)
True
isPrime(9)
False
isPrime(21)
False
```

5. *Euler's Method:* Numerical integration is very important for solving ordinary differential equations which cannot be solved analytically. In such cases, an approximation will have to do, and there are many different algorithms that achieve different levels of accuracy. One of the simplest methods to understand and implement is Eulers method, which is essentially a first order approximation. One step using Eulers method from t_n to t_{n+1} is given by

   $$y(t_{n+1}) = y(t_n) + hf(t_n, y(t_n))$$

   where h is the step size and $\frac{dy}{dt} = f(t, y)$. Now, write a function `approx_ode()` by implementing the Euler's method with step size, $h = 0.1$, to find the approximate values of $y(t)$ up to 3 decimal places for the following initial value problem (IVP):

   $$\frac{dy}{dt} = 3 + e^{-t} - \frac{1}{2}y, \quad y(0) = 1$$

   from $t = 0$ to $t = 5$ at a time interval of h. Note that the above IVP can also be solved exactly by the integrating factor method.

   Test Cases:

   Test case 1
   Input: t = 3
   Output: y(3) = 5.292

   Test case 2
   Input: t = 4
   Output: y(4) = 5.601

   Test case 3
   Input: t = 5
   Output: y(5) = 5.771

Problems: Exercises

1. *Functions: types:* Write a function named `mayIgnore()`. This function takes one argument that could be an integer, a float, a string, a complex, or of any other valid type in Python. The function adds 1 to the input and returns the sum but only if the input is an integer. If the input is not an integer then the function returns the value `None`. Use the function you have written to print the values of `mayIgnore(1)`, `mayIgnore(1.0)`, `mayIgnore(1+2j)`, and `mayIgnore(‘‘Hello")`. [Note that in order to solve this problem you need to use the `types` module in Python. Search the Internet to find how to use the `types` module.]

2. *Functions: reverse:* The `reverse()` member function of a list reverses the list. Write a function `myReverse`, which takes a list as input and returns the reverse of the list, without changing the original list. For example, if the input list is [5, -2, 15, 4] then the function must return the list [4, 15, -2, 5]. Use loops and do not use any built-in function to reverse a list.

3. *Functions and loops: approximation of* π Write a function that takes an input n and returns the value of π via:

   $$\frac{1}{\pi} = \frac{2\sqrt{2}}{9801} \sum_{k=0}^{n} \frac{(4k)!\ (1103 + 26390k)}{(k!)^4\ 396^{4k}}$$

4. *Loops: GCD:* Write a function that takes two positive integers and returns their greatest common divisor. (Hint: one naive solution is to try all possible divisors and find the largest one. But there are better methods to do it — `http://en.wikipedia.org/wiki/Greatest_common_divisor`).

5. Simpsons's rule for numerical integration: One of the methods for numerical integration is to use the Simpson's rule. Sometimes this method produces more accurate answers than the trapezoidal rule. Given a single valued and continuous function $f(x)$, the following formula corresponds to the Simpson's rule and works well for small values of h even when the function is not smooth over the interval of integration.

   $$\int_a^b f(x)dx \approx \frac{h}{3}\left[f(x_0) + 2\sum_{j=1}^{n/2-1} f(x_{2j}) + 4\sum_{j=1}^{n/2} f(x_{2j-1}) + f(x_n)\right]$$

   Here $h = (b-a)/n$, $x_0 = a$, $x_n = b$, and $x_j = a + jh$, for $j = 0, 1, 2...n-1, n$.

   Write a function named `simpsonsRule()`, that takes a function f(), integer $n > 1$, limits a and b as inputs and returns an approximation to $\int_a^b f(x)dx$. Use your function to approximate each of the following integrals. To understand the accuracy of the numerical

method used in this problem, manually compute the error in the approximation obtained numerically. For each integral, compare the errors obtained when using the Simpson's rule with those obtained using the n-point trapezoidal rule.

$\int_1^3 x^2 dx$ $\int_0^\pi sin(x) dx$ $\int_{-1}^1 e^{-x} dx$

Note: For those of you who want to learn more about numerical integration read the following article on Simpson's rule `http://en.wikipedia.org/wiki/Simpson%27s_rule`.

Test Cases:

Note: All outputs below are rounded to two decimal digits.

Test case 1

Input: $f = x^2$, $n = 1000, a = 1, b = 3$
Output: 8.67

Test case 2

Input: $f = sin(x)$, $n = 1000, a = 0$, $b{=}\pi$
Output: 2.0

Test case 3

Input: $f = e^{-x}$, $n = 1000, a = -1, b = 1$
Output: 2.35

End of Problem Set 3.