CS 3100 – Models of Computation – Fall 2011
This assignment is worth 4% of the total points for assignments
100 points total

August 23, 2011

Assignment 1, Handed out: August 23, 2010, Due: 9/1/11 - at or before 23:59:59 hours

Each question carries the same name as the file-name under which it should be submitted. See examples below under each question. Handin each question's file under **asg1** by **9/1 midnight**.

We will be automatically grading your submissions. Basically, we will use a Python script that iterates through all your folders, looks for a file with the prescribed name, and executes the file. *Therefore if you don't conform to our specification and give your file(s) or function(s) non-standard names (other names than prescribed below), you will lose your points!*

Grading **this assignment** out of 100 points, I provide distributions for the individual questions below.

test.py (**5 points**) Here is a python program you must all submit, just to test our automatic grading system.

- Enclose the following lines in a file called `test.py`

```
#!/usr/local/bin/python3

def test():
    print('hello')

if __name__ == "__main__":
    test()
```

- Execute the Unix command `chmod a+x test.py`
- Now type `./test.py`
- You must see `hello` printed
- Now handin `test.py` under `asg1`.
- **In general, for each program you are asked to submit**, place the program in the prescribed file. Then have `if __name__ == "__main__":` as the penultimate (last but one) line. As your last line, invoke the entry-point function ("top-level function") that runs your code. In this example, the top-level function is also called `test`—a practice I require.
- Thus, to recap, if a solution is in a file `foo.py`, have a function `foo()` in this file, taking the necessary arguments. Do not expect `foo` to take more inputs (besides these arguments) from a keyboard. If more inputs are needed, it must take it from a file situated in the same directory as `foo.py`.

LecSummary.txt (**10 points**) Read my lecture notes `http://www.eng.utah.edu/~cs3100/lectures/l1/notes1.pdf`. Out of the points made there, choose one that appeals to you the most. Write a one-page elaboration of this point, obtaining suitable supplementary material (*e.g.*, from the web). Submit as **LecSummary.txt**.

TeachingMath.txt (**10 points**) Watch the video at link `http://www.youtube.com/watch?v=60OVlfAUPJg&feature=player_embeddedD` and write a short paragraph of your thoughts. In a sense, this class represents a small step in moving CS 3100 toward a programming based class.

PythonEx1.py (**10 points**) Write a Python program to do these tasks:

- Determine the ordinal position of `'z'` using the `ord` function.
- Refer to it by a local variable `ordz`.
- Print `ordz` as a natural number.
- Follow this with a `for` loop that prints all 26 lower-case characters from `z` going backwards all in one line, separated by a blank.

Your solution must use the call `range(26)` to iterate over, determine the ordinal values of each preceding character from `ordz`.

Here is what your solution should look like. Your file ought to contain one function as follows, plus whatever else is necessary to support `PythonEx1`. We will be automatically grading most

```
def PythonEx1():
     ordz = the ordinal of z
     print ordz
o     for ch in range(...):
         print(..., end = ' ')
     print('')

if __name__ == "__main__":
    PythonEx1()
```

Your output should look like this:

```
122
z y x w v u t s r q p o n m l k j i h g f e d c b a
```

Late Addition (do it if you can, please): Please output your result into a file `PythonEx1.out`.

Palindrome.py (**10 points**) Write a Python function that accepts a string `str` as argument, and makes a palindrome out of it by mirroring `str` and appending the mirrored string at the end of `str`. Test it on four different inputs.

Example:

```
Given abcd, produce abcddcba
```

Your submission ought to be in file `Palindrome.py` and contain one function `Palindrome(s)` plus whatever else that supports this function (includes, etc—you may have nothing else, for this simple an example, but in general you will have). Your submission will look like this:

```
def Palindrome():
    ...code...

def run_Palindrome():
    Palindrome(..your input 1..)
    Palindrome(..your input 2..)
    Palindrome(..your input 3..)
    Palindrome(..your input 4..)

if __name__ == "__main__":
    run_Palindrome()
```

FURTHER SPECIFICATION:

- Make your program print the outputs, one output per line, into `Palindrome.out`
- The inputs must be as follows

  ```
  ..your input 1.. must be "abca13"
  ..your input 2.. must be "(()()"
  ..your input 3.. must be ""
  ..your input 4.. must be "z"
  ```

SuffixClosure.py (**10 points**) Someone claims that the following function generates the suffix-closure of a given string `s`:

```
def suffclosure(s):
    return { s[i:len(s):] for i in range(len(s)+1) }
```

Test this function out by applying it to four different strings. Your submission will look like this:

```
def suffclosure(s):
    return { s[i:len(s):] for i in range(len(s)+1) }

def run_suffclosure():
    suffclosure(..your input 1..)
    suffclosure(..your input 2..)
    suffclosure(..your input 3..)
    suffclosure(..your input 4..)

if __name__ == "__main__":
    run_suffclosure()
```

FURTHER SPECIFICATION:

- Make your program print the outputs, one output per line, into `SuffixClosure.out`
- The inputs must be as follows

  ```
  ..your input 1.. ""
  ..your input 2.. "a"
  ..your input 3.. "(((())"
  ..your input 4.. ";ajsdlkfjalsj dd"
  ```

PrefixClosure.py (**20 points**) Write a function similar to Suffclosure, but for computing the prefix-closure.

FURTHER SPECIFICATION:

- Make your program print the outputs, one output per line, into `PrefixClosure.out`
- The inputs must be as follows

```
..your input 1.. ""
..your input 2.. "a"
..your input 3.. "((((()"
..your input 4.. ";ajsdlkfjalsj dd"
```

FacList.py (**25 points**) Write a function that, given N, generates the list of factorials of all numbers from 1 to N. For example, given 5, your function must return

```
[1, 2, 6, 24, 120]
```

Your solution must employ a list comprehension to generate the final list. Test it for four different values of N (try N up to 50 or 100 and see what happens).

Your submission will look like this:

```
def FacList(N):
    ...code...

def run_FacList():
    FacList(..input 1..)
    FacList(..input 2..)
    FacList(..input 3..)
    FacList(..input 4..)

if __name__ == "__main__":
    run_FacList()
```

FURTHER SPECIFICATION:

- Make your program print the outputs, one output per line, into `FacList.out`
- The inputs must be as follows

```
The ..your input 1.. part must be 10 for the first input
..your input 2..  = 1
..your input 3..  = 2
..your input 4..  = 50
```