Homework 1

Due 4:00pm, Wednesday, February 6, 2019

Installation and Handin

Homework Setup. For most homework assignments, there will be support files that you will need to complete the assignment. These can be copied to your home directory by using the `cs4_install` command in a Brown CS Terminal window. For this homework, type in terminal:

```
cs4_install hw01
```

There should now be a `hw01` folder within your homeworks directory. Using Terminal, you can move into the `hw01` folder with the cd command:

```
cd ~/course/cs0040/homeworks/hw01
```

Homework hand-in. When you have finished the homework, be sure to turn in all the files requested and that they are named exactly as specified, including spelling and case. To submit the files, from a Brown CS Terminal window in your `~/course/cs0040/homeworks/hw01` directory, type:

```
cs4_handin hw01
```

The entire contents of your `~/course/cs0040/homeworks/hw01` directory will be handed in. Check for a confirmation email to ensure that your assignment was correctly submitted using the `cs4_handin` command. You can re-submit this assignment using the `cs4_handin` command at any time, but be careful, as only your most recent submission will be graded.

Part I: Strings and Lists (10)

This homework is divided into five parts. Make sure to pay attention to which file your answers for each part go in as you read the assignment specifications below.

Problem 1.1a) String Indexing and Slicing (5)

Open `hw01_1.py` in Atom. You are provided with the three strings and the following stencil code:

```
b = "brown"
u = "university"
a = "awesome"
# Example: Create the string "best"
answer_ex1 = b[0] + u[4:9:2]
print(answer_ex1)

# Create the string 'brownies'
answer1 = ''
print(answer1)
# Create the string 'emotion'
answer2 = ''
print(answer2)
```

Following the example, update `hw01_1.py` to create the requested strings using `b`, `u` and `a` and whichever of the following string operations you deem appropriate (hint: you don't have to use all of them!):

- string indexing (e.g., u[1])
- string slicing (e.g., u[1:3]),
- string skip-slicing (e.g., t[3:1:-1])
- string concatenation (e.g., b+u)
- string repetition (e.g., 2*t)

We encourage you to try and do this with as few string operations as possible. Be sure to save your updated answers back to `hw01_1.py`, and use then the following command:

```
python3 hw01_1.py
```

to try out your solutions. You should see the correct strings printed out. If they are not correct, double check your work and make some edits to ensure all the correct strings are printing!

Problem 1.1b) List Indexing and Slicing (5)

For the second part of this problem, you are provided with the following three lists and the following stencil code:

```
e = [2, 4, 6, 8]
o = [1, 3, 5, 7, 9]
c = [0,10]
# Example: Create the list [1, 4, 6]
answer_ex2 = [o[0]] + e[1:3]
print(answer_ex2)

# Create the list [0, 1, 5, 9, 10]
answer5 = []
print(answer5)
# Create the list [2, 6, 9, 5, 1]
answer6 = []
print(answer6)
```

Following the example, create the requested lists using whichever of the following string operations you deem appropriate (hint: you don't have to use all of them!):

- list indexing (e.g. e[1])
- list slicing (e.g. o[1:3])
- list skip-slicing (e.g. e[3:1:-1])
- list repetition (e.g. 2*c)
- list concatenation (e.g. [e[1]]+c)
- list construction (e.g. [e[0], c[1]])

Again, try to use as few list operations as possible. All answers should also be saved to `hw01_1.py`. Again, use the `python3 hw01_1.py` command to check your answers!

Part II: Tracing Program Execution (20)

It's very important to know the flow of a program during its execution and how it affects the program's state (i.e. the value of all of its variables). By default, a program's statements are executed sequentially, from top to bottom. The value of a variable in a program changes as it is assigned different values during the execution of a program.

Below is a simple Python program. Here, `b` is first set to 5, and then updated to a new value of 8, which is its value for the rest of the program.

```
a = 15
b = 5
c = a + b
b = b + 3
a = b // 4
c = b ** 2
```

Problem 1.2a) Variable Reassignment (10)

The file `hw01_2a.py` contains a copy of the code above embedded in the table below. Fill in the table inside the file to show the values of the variables as each line of code is executed. The first two rows of the table are filled in for you. Once you are done, list the final values of `a`, `b`, and `c`.

```
'''
Complete the following table to show all variable changes

    line of code   | a  | b  | c  |
    -------------------------------
    a = 15         | 15 |    |    |
    b = 5          | 15 | 5  |    |
    c = a + b      |    |    |    |
    b = b + 3      |    |    |    |
    a = b // 4     |    |    |    |
    c = b ** 2     |    |    |    |

Q: What are the final value of a, b and c?
A: a =   , b =    and c =
'''
```

Problem 1.2b) Understanding Variable Scope (10)

Your goal is to:

- Walk through the code and identify which variables are local, and which variables are global. Some are used as both local and global variables in the code!
- Fill out the table at the bottom of the file, by designating each variable L (local only), G (global only), or B (both)

Part III: Debugging & Conditionals (10)

The `hw01_3.py` file contains the following code:

```
def decide(temp):
    """
    Print a decision based on an input temperature in Fahrenheit
    """
    if temp > 80:
        print('The temperature is','temp', ', so we will go to Newport.')
    if temp > 55 and temp <= 80:
        print('The temperature is','temp', ', so we will go to Boston.')
    if temp > 40 and temp <= 55
    print('The temperature is','temp,' ', so we will go to Eastside.')
    if temp > 32 and temp <= 40:
        print('The temperature is','temp', ', so we will go to the Avon.')
    if temp <= 32:
        print('The temperature is', temp, ', so we will stay in.'

decide(85)
decide(60)
decide(45)
decide(36)
decide(10)
```

The `decide` function is supposed to take as input a number indicating the temperature and print out a decision for your weekend plan. In its current state, the function contains several errors that prevent it from running properly.

To introduce you to the process of debugging, **your first task is to fix these syntax errors.** Syntax errors are simply typos that prevent the code from running (a missing parenthesis, incorrectly placed quotation mark, etc.). Once these errors are fixed, you should be able to run the given code and see the results. After each call to the `decide` function it should print:

```
The temperature is <XX> <decision>
```

where <XX> is the input temperature in Fahrenheit and <decision> is the plan for that temperature based on the rules of the function. For this bug fixing step, do not change how these are currently defined (just get them to work). Once you think you're done, you can try running the program with the `python3` command:

```
python3 hw01_3.py
```

Hints:

- If your program fails to run due to a syntax error, the error messages that are produced in the terminal can sometimes give you clues as to why your code is incorrect. In particular, the error messages will print out the line number on which your program is failing.
- A majority of the errors are related to missing parentheses, quotation marks or colons.
- Note the difference between `'temp'` and `temp`.

Part IV: Functions and Function Calls (30)

In this part we are going to write some simple functions. Write your answers to all parts of this problem in `hw01_4.py`. For each part, make sure you:

- **Include a docstring** right after the declaration of the function that describes what the function is doing. See the `decide` function in Part III for an example of this (docstrings are denoted by triple quotes `"""`).
- Do not use any built-in Python functions that we have not yet discussed in class.
- Make sure all function names and input variables are *exactly* as we have given them to you. Changing these names will make it difficult for us to test what you have written.
- **Return** the requested output, printing is optional!

Problem 1.4a) Functions with Numeric Inputs (15)

square(x)

Write a function named `square(x)`, that takes a number x as input and returns the square of its input. For example:

```
>>> square(2)
4
```

root(x)

Write a function named `root(x)`, that takes a number x as input and returns the square root of its input. For example:

```
>>> root(2)
1.41421356...
```

Hint: how can you express a root with arithmetic operators we have learned?

distance(x1, y1, x2, y2)

Write a function `distance(x1, y1, x2, y2)` that takes four inputs that specify the x- and y-coordinates of two points (x1, y1) and (x2, y2), and returns the distance between those two points. Sample input and output is shown below. Use the distance formula:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Note: Implement your distance function using the `square(x)` and `root(x)` functions you just wrote.

```
>>> distance(2, 3, 5, 7)
5.0
```

Problem 1.4b) Functions with Lists and Strings (15)

mid_element(numlist)

Write a function called `mid_element(numlist)` that takes as input a non-empty list of numbers and returns:

- the element if there is only one element in `numlist`
- the middle element if there is an odd number of elements in `numlist`
- the average of the two middle elements if there is an even number of elements in `numlist`

Note: The only built-in Python function you are allowed to use is `len()`.

```
>>> mid_element([1])
1
>>> mid_element([1, 2, 3, 4, 5])
3
>>> mid_element([1, 2, 3, 4])
2.5
```

Hint: `/` is used for float division and `//` is used for integer division. Make sure you are using the correct one!

smaller_len(s1, s2)

Write a function called `smaller_len(s1, s2)` that takes as input two different strings, and returns the length of the smaller string.

Note: You are not allowed to use the Python built-in `min()` function. If the strings are the same length, return s1.

```
>>> smaller_len("hello", "hi")
2
```

Part V: List Comprehensions (30)

Place your answers for Part III in the file named `hw01_5.py`.

Problem 1.5a) List Comprehension Puzzles

You will see that `hw01_5.py` includes several incomplete list comprehensions. Complete them by filling in the blanks in the file to produce the results specified below.

1. `[5, 6, 7, 8, 9]`.
2. `['going', 'eating', 'reading']`.
3. `['o', 'd', 'w', 's', '?']`.
4. `[5, 10, 15, 20]`. Note that you need to fill in two blanks for this one: the expression before the `for`, and the expression after the `if`.

Problem 1.5b) Functions Using List Comprehensions

In this problem you will write two functions.

Write a function called `longer_than(n, wordlist)` that takes as inputs an integer `n` and a list of strings `wordlist`, and that *uses a list comprehension* to construct and *return* a list consisting of all words from `wordlist` that are longer than `n`. (Hint: your list comprehension will need an `if` clause.) For example:

```
>>> longer_than(3, ['only', 'recursion', 'on', 'the', 'brain'])
```

```
['only', 'recursion', 'brain']
>>> babynames = ['Alexander', 'Katherine', 'Ben', 'Annabel']
>>> longer_than(6, babynames)
['Alexander', 'Katherine', 'Annabel']
>>> longer_than(7, babynames)
['Alexander', 'Katherine']
>>> longer_than(10, babynames)
[]
```

Write a function called `most_vowels(wordlist)` that takes a list of strings `wordlist` and returns the string in the list with the most vowels. You may assume that the strings only contain lowercase letters. For example:

```
>>> most_vowels(['Buddy', 'Oreo', 'Max', 'Toto'])
'Oreo'
>>> most_vowels(['puppies', 'are', 'the', 'greatest'])
'puppies'
```

The function that you write should use a `num_vowels` function as a helper function. We strongly encourage you to use the list-of-lists technique that we discussed in lecture, but you may use some other technique if you prefer.

Note: When two or more words are tied for the most vowels, it doesn't matter which one you return as long as the one returned is part of the subset that is tied for most vowels.

Please let us know if you find any mistakes, inconsistencies, or confusing language in this document or have any concerns about this and any other CS4 document by posting on Piazza or filling out our anonymous feedback form.