15-110: Principles of Computing, Spring 2018

Lab 5 – Thursday, February 15

Goals

- Practice debugging code

Even though it may seem easy to come up with an algorithm to solve a problem, you may end up with a number of errors when you translate the algorithm to a program. Learning how to debug efficiently (rather than randomly hacking at your code) will help you write correct programs more quickly. The following are some steps to remember as you debug:

1. Run your code with `python` and check for syntax errors. (Not fulfilling the requirements for the language format will result in syntax errors and your code will immediately fail when Python loads.)
2. Run your functions on specific inputs and check if you get your expected output.
3. Read through your code and observe what the function is doing at each step. See if this matches your expectation of what it should do.
4. Put `print` statements at logical points in your code to print the contents of a specific variable or list. Check if the printed outputs are what you expect your program to be doing. Do you see a long series of outputs that never seems to end? Maybe you have an infinite loop.

When Python crashes loading your file or running your function, **reading the printed error message is helpful**. The message indicates the line number at which the error is happening as well.

Part 1 : Reading the Error Message [TA Demonstrations]

1.1 Syntax Error and Logical Error

The following table describes two different types of errors that you may encounter when writing code:

Syntax Error	Logical Error
• Fails immediately when Python loads • Occurs when not satisfying the requirements for the language format	• Code compiles, but… • Python gives the wrong answer • Does not behave the way it is intended to

`def add_elements(numlist):`
`  result = 0`
`  for i in range(len(numlist))`
`  # missing a colon!`
`  result = result + numlist[i]`
`  return result`
	`def add_elements(numlist):`
	`  result = 0`
	`  for i in range(len(numlist)):`
	`  result = result + i`
	`  # adding index, not element`
	`  return result`

`-bash-4.2$ python3 –i add_elements.py`
`File "add_elements.py", line 3`
`  for i in range(len(numlist))`
`  ^`
`SyntaxError: invalid syntax`
	`-bash-4.2$ python3 –i add_elements.py`
	`>>> add_elements([10, 20, 30, 40])`
	`6`
	`>>> # what did we expect?`

You should generate information or clues about the behavior of your program before asking for help. Looking at the clues, programmers can often figure out what is wrong on their own. When you write code and errors happen, put on your detective cap and start to look for clues! The more you do this, the better you will be at writing programs!

Part 2 : Student Activities

Create a file `answers.txt` using gedit. For each of the programs below, find the bug(s) and place your **answers to the following questions for each program** in `answers.txt`.

a. If you got a syntax error, how did you figure out what was wrong?
b. What arguments did you use to demonstrate any logical error?
c. What output(s) did you expect with the argument(s) used above and what did you get instead?
d. How did the steps above help you find the logical error?

You may put N/A as your answer if the question does not apply. Also, you should **correct the code** after running your debugging test(s) and save the file as the specified name. To save time, copy and paste the original code into your editor before modifying it.

1. The function `contains(element_list, key)` should return `True` if `element_list` contains `key` and `False` otherwise.

```
def contains(element_list, key):
    for item in element_list:
        if item == key:
            print(True)
    print(False)
```

Create a file `contains.py` and copy and paste the code above. Determine the bug(s), write the answers to the questions above in `answers.txt`, and save the file after correcting the code.

2. The function `get_index(element_list, key)` should return the first index of `key` in `element_list`. It should return -1 if `key` is not in `element_list`.

```
def get_index(element_list, key):
    for i in range(0, len(element_list)):
        if element_list(i) == key:
            return i
        else:
            return -1
```

Create a file `get_index.py` and copy and paste the code above. Determine the bug(s), write the answers to the questions above in `answers.txt`, and save the file after correcting the code.

3. `pow(base, exp)` is the power (exponentiation) function which is defined recursively. It should raise `base` to `exp`. In other words, it is equivalent to `base ** exp`. However, the function must work recursively; you cannot just correct it by returning `base ** exp` to solve this problem.

To compute $base^{exp}$ recursively, we note that $base^{exp} = base \times base^{exp-1}$ with the special case that $base^{exp} = base$ if $exp = 1$.

```
def pow(base, exp):
    return base * pow(base, exp - 1)
    if exp == 1:
        return base
```

Create a file `pow.py` and copy and paste the code above. Determine the bug(s), write the answers to the questions above in `answers.txt`, and save the file after correcting the code.

4. The function `square_evens(numlist)` square all even numbers in `numlist` and print the resulting list.

```
def square_evens(numlist):
    for num in numlist:
        if num % 2 == 0:
            num = num ** 2
    print(numlist)
```

Create a file `square_evens.py` and copy and paste the code above. Determine the bug(s), write the answers to the questions above in `answers.txt`, and save the file after correcting the code.

5. The function `list_mult(numlist, multiplier)` should multiply each of the numbers in `numlist` by `multiplier`. The function `test_mult(numlist)` should use `list_mult` to create and print two different modifications of the given list: `two_mult` should be a list made up of all of the original list elements multiplied by 2 and `five_mult` should be a list made up of all of the original list elements multiplied by 5.

```
def list_mult(numlist, multiplier):
    for i in range(0, len(numlist)):
        numlist[i] = numlist[i] * multiplier
    return numlist

def test_mult(numlist):
    two_mult = list_mult(numlist, 2)
    five_mult = list_mult(numlist, 5)
    print(two_mult)
    print(five_mult)
```

Create a file `list_mult.py` and copy and paste the code above. Determine the bug(s), write the answers to the questions above in `answers.txt`, and save the file after correcting the code.

Submission

When you finish the lab, you should be inside the `lab5` folder, which is inside the `private/15110` directory. When you type '`ls`' and press the `Enter` key, you should see the following files: **`answers.txt`**, **`contains.py`**, **`get_index.py`**, **`pow.py`**, **`square_evens.py`**, and **`list_mult.py`**. Once you see all files, please type '`cd ..`' and press the `Enter` key. Then, zip your `lab4` folder by typing '`zip –r lab5.zip lab5`'. Please submit the zipped file `lab5.zip` on Autolab under lab 5.