Py_Project_1_due_1020

October 20, 2024

1 Python Assignment 1

Due: 10/20/2024 11:59:59 PM

Submit this notebook file and upload it to your git repo. Include a link to your git repo while submitting this file to Brightspace.

You are not allowed to use chatgpt or any LLM in this assignment. Cooperation is allowed and encouraged; however, you must write up and submit your own work. If you cooperate with others, you must list their names here.

Total points: 100 (equivalent to 20 points after conversion)

1.1 0. Data Types, Structures, Indexing, and Slicing

Read the lecutre note as well as this page and this page. Complete the following questions.

(5 points) Give some examples (at least 2) of mutable objects in Python.

Answer: Lists, dictionaries, and sets

(5 points) Give some examples (at least 2) of immutable objects in Python.

Answer: Int, float, bool, string

(10 points) State the differences between assignment, shallow copy, and deep copy. Any reasonable answer will get full credit.

Answer: Assignment does not actually make a copy, but rather assigning two variables to one thing. In shallow copy, the changes of elements in the new object may change the elements in the original object if referred. However, in deep copy, it creates a brand new object, and the original object will never change with the changes of the new object.

(10 points) Using string methods find() and rfind(), find the index of the first digit (i.e. 0) and the index of the last digit (5) in the string below. Then, using string slicing, extract the number (i.e. 0.595595) from this string and convert the extracted value to a floating point number. Print out this float to the console.

```
[1]: text = "F-DGFDDSFGFD-CFDSHdstgfdfe: tfsd  aaa bdsf 0.595595 fdsgfdbc"

     #FIND THE INDICES
     index_of_zero = text.find('0')
     index_of_last_five = text.rfind('5')

     #EXTRACT AND SLICING
     extract_num = text[index_of_zero:index_of_last_five+1]
     float_num = float(extract_num)

     #PRINTING
     print(float_num)
```

```
0.595595
```

1.2 1. Conditional Statements

(10 points) Translate the following MATLAB code into Python using `if-elif-else`.

```
n = input('Enter a number: ');

switch n
    case -1
        disp('negative one')
    case 0
        disp('zero')
    case 1
        disp('positive one')
    otherwise
        disp('other value')
end
```

```
[2]: n = int(input('Enter a number:'))

     if n == -1:
         print('negative one')
     elif n == 0:
         print('zero')
     elif n == 1:
         print('positive one')
     else:
         print('other value')
```

```
Enter a number:8
other value
```

1.3 2. While Loops

(10 points) Write a Python program that calculates the factorial of a given number (you can assume an positive integer) using a while loop. Display the result to the console.

[3]:
```python
def factorial(n):
    if n == 0:
        return 1
    else:
        return n*factorial(n-1)

print(factorial(17),factorial(13))

# test cases
# n = 17, Output: 355687428096000
# n = 13, Output: 6227020800
```

```
355687428096000 6227020800
```

1.4 3. Functions

(10 points) Write a function that determines if a number is prime.

[4]:
```python
def is_prime(n):
    if n == 1:
        return False
    else:
        for i in range(2, n):
            # Check if the number has a divisor
            if (n%i == 0):
                return False
        else:
            return True



# test cases
print(is_prime(2))      # Output: True
print(is_prime(10))     # Output: False
print(is_prime(17))     # Output: True
```

```
True
False
True
```

1.5 4. List Comprehension

(10 points) Write a Python function called `prime_factors` that takes a relatively small integer as input and returns a list of its distinct prime factors. The function should use list comprehension to generate the list of distinct prime factors. You may use the function is_prime() defined in Q3.

[5]:
```python
def prime_factors(n):
    factors = []
    for i in range(1, n + 1):
        # If i is a factor
        if n % i == 0:

            # If i is prime
            if is_prime(i):
                factors.append(i)

    return factors


# test cases
print(prime_factors(30))  # Output: [2, 3, 5]
print(prime_factors(56))  # Output: [2, 7]
print(prime_factors(198))  # Output: [2, 3, 11]
print(prime_factors(464))  # Output: [2, 29]
```

```
[2, 3, 5]
[2, 7]
[2, 3, 11]
[2, 29]
```

1.6 5. Recursive Functions

(25 points) Write a function that uses recursion to generate the nth row of the Pascal's triangle.

```
              1
           1     1
        1     2     1
      1    3     3     1
   1    4     6     4    1
1     5    10    10    5    1
...
...
...
```

[6]:
```
def pascal(n):
    # Base Case
    if n == 1:
        return [1]
    else:
        # Obtain the last row with recursion
        last_row = pascal(n-1)

        # Initialization
        row = [1]

        # Operations
        for i in range(len(last_row)-1):
            row.append(last_row[i]+last_row[i+1])

        # The ending 1
        row += [1]
        return row

# test case
print(pascal(6))  # output [1, 5, 10, 10, 5, 1]
```

```
[1, 5, 10, 10, 5, 1]
```

Rewrite the above function using only `while` or `for` loops instead of recursion.

[7]:
```
def pascal_by_loops(n):
    # First row
    row = [1]

    for i in range(1, n):
        # Start with 1
        new_row = [1]

        for j in range(len(row) - 1):
            new_row.append(row[j] + row[j + 1])

        # The ending 1
        new_row.append(1)

        # Complete recursion
        row = new_row

    return row

print(pascal_by_loops(6))
```

```
[1, 5, 10, 10, 5, 1]
```

---

5 points) good coding practice and properly submitting your work to GitHub.