Python Assignment 1

⏱⏱⏱Due: 10/19/2025 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)

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, sets

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

Answer: int, float, str

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

Answer: Assignment creates a new reference to the object while shallow copy creates a new object referencing the same stuff (inner objects).

Deep copy copies everything, including the inner objects, so it is more than a shallow copy because it is referencing copies.

(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.

```
text = "F-DGFDDSFGFD-CFDSHdstgfdfe: tfsd  aaa bdsf 0.595595 fdsgfdbc";

# find first and last digit
first_digit = text.find('0')
last_digit = text.rfind('5')

# slice to extract
number_str = text[first_digit:last_digit+1]

# convert to float
number = float(number_str)

# print
print(number)
```

```
0.595595
```

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
```

```
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: 1
positive one
```

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.

```
n = int(input("Enter a positive integer:"))

factorial = 1
i = 1

while i <= n:
    factorial *= i
    i += 1

print("The factorial of", n, "is", factorial)



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

```
Enter a positive integer:17
The factorial of 17 is 355687428096000
```

3. Functions

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

```
def is_prime(n):
  # less than 2 is not prime
  if n < 2:
    return False

  # divisibility
  i = 2
  while i * i <= n:
    if n % i == 0: # modulus = 0
      return False
    i += 1

  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
```

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.

```
def prime_factors(n):
  factors = [i for i in range(2, n+1) if n % i == 0 and is_prime(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]
```

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
...
...
...
```

```
def pascal(n):
  # first row 1
  if n == 1:
    return [1]

  # recursively get prev row
  prev_row = pascal(n-1)

  # current based on prev
  curr_row = [1]
  for i in range(1, n-1):  # elements up to n-2
    curr_row.append(prev_row[i-1] + prev_row[i]) # sum adjacent elements
  curr_row.append(1)

  return curr_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.

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

  for i in range(1, n):
    new_row = [1]
    for i in range(1, len(row)): # length
      new_row.append(row[i-1] + row[i]) # sum adjacent elements
    new_row.append(1)
    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.