Home Work 5

July 3, 2024

Dylan Liesenfelt

1 Section

Write a function named count_primes that returns the number of prime numbers that exist up to and including a given number. Demonstrate how to call the function and display the output.

[ ]:
```
def check_if_prime(number): # This function checks if our number is prime
    if number < 2:  # first prime number is 2 so no need to check before this
        return False
    for i in range(2, number): # counting through 2 to our number
        if number % i == 0: # If our number is divisible by another number␣
 ↪besides 1 our itself (for loop stops just counting at number before our␣
 ↪number), then it is not a prime
            return False
    return True # if all other conditions are not true than our number is a␣
 ↪prime, return value of true to function

def count_primes(number): # This is our main function and counts all of our␣
 ↪primes
    primes = [] # List to contain all numbers we find that are prime
    for i in range(2, number + 1): # Once again 2 is first prime so we start␣
 ↪there, counts up to our number this time
        if check_if_prime(i) == True: # Calls our function to check if current␣
 ↪number being counted is prime
            primes.append(i) # If it is add it to our prime list
    print(f'Prime numbers up to {number}: {primes}') # Once were done counting␣
 ↪print the results

num = int(input('Enter a number to count the primes up to that number: '))
count_primes(num) # Call our function passing in the user input as the param
```

```
Prime numbers up to 29: [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
```

2 Section

Convert the following functions into lambda expressions. Test both the original and lambda versions.

[ ]:
```python
# def square(num):
#   return num ** 2

square = lambda num: num ** 2
print(square(5))
```

```
25
```

[ ]:
```python
# def add(x, y):
#   return x + y

add = lambda x, y: x + y
print(add(10,15))
```

```
25
```

[ ]:
```python
# def is_even(num):
#    return num % 2 == 0

is_even = lambda num: num % 2 == 0
print(is_even(5))
```

```
False
```

[ ]:
```python
# def concatenate(str1, str2):
#   return str1 + str2

concatenate = lambda str1, str2: str1 + str2
print(concatenate('Hello', ' World'))
```

```
Hello World
```

[ ]:
```python
# def string_length(s):
#   return len(s)
string_length = lambda s : len(s)
print(string_length(';iolksdfhfg;kjlerhg;jkdsh;k'))
```

```
27
```

[ ]:
```python
# def reverse(a):
#   a_reversed = a[::-1]
#   return a_reversed

reverse = lambda a : a[::-1]
print(reverse('Hello World'))
```

```
dlroW olleH
```

3 Section

Convert these conditional structures to ternary operations and test both versions.

```
[ ]: # if num > 0:
     #   return "Positive"
     # elif num == 0:
     #   return "Zero"
     # else:
     #   return "Negative"

     num = -5

     result = 'Positive' if num > 0 else 'Zero' if num == 0 else 'Negative'

     print(result)
```

```
Negative
```

```
[ ]: # if lst:
     #   return "Not empty"
     # else:
     #   return "Empty"

     lst = ['a',1]

     result = 'Not empty' if lst else 'Empty'

     print (result)
```

```
Not empty
```

```
[ ]: # if age >= 18:
     #   return "Adult"
     # else:
     #   return "Minor"
     age = 47

     result = 'Adult' if age >= 18 else 'Minor'

     print(result)
```

```
Adult
```

```
[ ]: # if num % 2 == 0:
     #   return "Even"
     # else:
     #   return "Odd"
```

```
num = 3456

result = 'Even' if num % 2 else 'Odd'
print(result)
```

```
Odd
```

4 Section

Provide code and test output for each:

[ ]:
```
# a) Create a function that uses several default values.
def severalDefaults(greet='Hello', name='Person', age='Unkown'):
    print(f'{greet}, {name} of {age} years old')

severalDefaults()
severalDefaults('Hey there', 'Joe', 25)
```

```
Hello, Person of Unkown years old
Hey there, Joe of 25 years old
```

[ ]:
```
# b) Create a function that uses positional arguments *args.

def nArgs(*args):
    for i,j in enumerate(args):
        print(f'Student Number: {i}, Name: {j}')

nArgs('Tom', 'Joe', 'Mike')
```

```
Student Number: 0, Name: Tom
Student Number: 1, Name: Joe
Student Number: 2, Name: Mike
```

[ ]:
```
# c) Create a function that uses keyword arguments **kwargs.
def kwarg(**kwargs):
    for key, value in kwargs.items():
        print(f'{key} : {value}')

kwarg(Name='Joe', ID = 'z1654896', Age = 25, Major = 'CS')
```

```
Name : Joe
ID : z1654896
Age : 25
Major : CS
```

[ ]:
```
# d) Create a function that combines *args, **kwargs, and default values.
def womboCombo(*args, length=1, **kwargs):
    for i,j in enumerate(args):
        for key, value in kwargs.items():
            print(f'{key * length} {value * (i ** j)}')

womboCombo(5, 10, x = 3, y = 2)
```

```
x 0
y 0
x 3
y 2
```

5 Section (Bonus)

Create a function that prints digits or characters as ASCII art, and then use it to write your zNumber or any other message.

```
[ ]: def ASCIIzNumber(zNum):
         ASCII_Dict = { # Dict that contains the characters we want and the ASCII␣
      ↪art associated to that char
             '0': [' 00000 ',
                   '000 000',
                   '00   00',
                   '00   00',
                   '000 000',
                   ' 00000 '],
             '1': ['   111 ',
                   ' 11111 ',
                   '   111 ',
                   '   111 ',
                   '   111 ',
                   ' 111111'],
             '2': [' 22222 ',
                   '22   22',
                   '    22 ',
                   '  22   ',
                   '22     ',
                   '2222222'],
             '3': [' 33333 ',
                   '3    33',
                   '   333 ',
                   '   333 ',
                   '3    33',
                   ' 33333 '],
             '4': ['   4444',
                   '  44 44',
                   ' 44  44',
                   '4444444',
                   '     44',
                   '     44'],
```

```python
        '5': ['555555 ',
              '55     ',
              '55555  ',
              '    55 ',
              '    55 ',
              '55555  '],
        '6': [' 66666 ',
              '66     ',
              '66666  ',
              '66  66 ',
              '66  66 ',
              ' 6666  '],
        '7': ['7777777',
              '    77 ',
              '   77  ',
              '  77   ',
              ' 77    ',
              '77     '],
        '8': [' 8888  ',
              '88  88 ',
              ' 8888  ',
              ' 8888  ',
              '88  88 ',
              ' 8888  '],
        '9': [' 99999 ',
              '99   9 ',
              ' 99999 ',
              '    99 ',
              '    99 ',
              ' 99999 '],
        'Z': ['      ',
              '      ',
              'ZZZZZZ',
              '   ZZ ',
              ' ZZ   ',
              'ZZZZZZ']
    }

    lines = ['' for _ in range(6)] # Init our list to have an empty list with space for each line of each characters art

    for i in zNum: # Loop through the characters in the znumber
        c = i.upper() # Force current char to be upper case
        if c in ASCII_Dict: # If our character is the art dict then ...
            charLines = ASCII_Dict[c]  # assign value to var
            for j in range(6): # loop through for each line
                lines[j] += charLines[j] + ' ' # add the lines for character
    for line in lines: #loop out all the lines in our list and print them line␣
↪by line
        print(line)

ASCIIzNumber("Z23688417") # Call our function by passing my zNumber
```

```
        22222   33333   66666   8888    8888      4444    111  7777777
       22   22 3    33 66      88  88  88  88    44 44  11111      77
ZZZZZZ     22     333  66666    8888    8888    44  44    111     77
   ZZ    22       333  66  66   8888    8888   4444444    111    77
 ZZ    22      3    33 66  66  88  88  88  88       44    111   77
ZZZZZZ 2222222  33333   6666    8888    8888        44  111111 77
```