EE2703 - Week 1

Hardik Gagrani, EE21B047

February 4, 2023

1 Document metadata

Problem statement: modify this document so that the author name reflects your name and roll number. Explain the changes you needed to make here. If you use other approaches such as LaTeX to generate the PDF, explain the differences between the notebook approach and what you have used.

#Explanation By opening Python Notebook , and replacing the author's name by my name and roll nunmber.

2 Basic Data Types

Here we have a series of small problems involving various basic data types in Python. You are required to complete the code where required, and give *brief* explanations of your answers. Remember that the documentation and explanation is as important as the answer.

For each of the following cells, first execute them, and then give a brief explanation of why the answer comes out to be the way it does. If there is an error during execution of the cell, explain how you fixed it. **Add a new cell of type Markdown with the explanation** after the corresponding cell. If you are using plain Python, add suitable comments after each line and explain this in the documentation (clearly you would be better off using Notebooks here).

2.1 Numerical types

[2]:
```
print(12 / 5)
```

```
2.4
```

In this line of code, normal divison takes place resulting in floating point number 2.4

[3]:
```
print(12 // 5)
```

```
2
```

In this line of code, integer divisor takes place (floor divison) which gives us the floor value 2.

[4]:
```
a=b=10
print(a,b,a/b)
```

```
10 10 1.0
```

Here we assign value of 10 to a and b simultaneously using '=' operator and thus printing (a, b, a/b) we get 10, 10 and 1.0 (because we are using normal division here)

2.2 Strings and related operations

[7]:
```
a = "Hello "
print(a)
```

```
Hello
```

Here, "a" is a string variable holding the value "Hello," leading to the output of the above result.

[8]:
```
print(a+b)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[8], line 1
----> 1 print(a+b)  # Output should contain "Hello 10"

NameError: name 'b' is not defined
```

This line of code is giving us an error because we are adding 2 variables of different data types and in python we can add 2 different variable only if they have the same data type. Here data type of a is string and on the other hand data type of b is int and so it is showing an error.

[9]:
```
b = "10 "
print(a+b)
```

```
Hello 10
```

To rectify the above error I initialised variable b again as 10 but this time as a string data type and hence we are getting the result "Hello 10" upon printing (a+b).

[16]:
```
print("-"*40)
print(f"{42:>41}")
print("*-"*20)
```

```
----------------------------------------
                                        42
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
```

To print "-" 40 times, I am using the function "* quantity" to print it for required number of times and similarly used this function only to print "*-" line of length 40 Nextly, using the function ":>" I am priting "42" right justified by 41 units

[13]:
```
print(f"The variable 'a' has the value {a} and 'b' has the value {b:>10}")
```

```
The variable 'a' has the value Hello  and 'b' has the value        10
```

Here the function {b:>10} shifts the value b or the value stored in b right wards by 10 spaces. Similarly, if we use {a:<10} it will make the value left shifted by 10 spaces.

```
[20]: Courses_List = [{"ID" : "EE2703", "Name" : "Applied Programming Lab"},
                      {"ID" : "EE2003", "Name" : "Computer Organisation"},
                      {"ID" : "EE5311", "Name" : "Digital IC Design"}]

      for i in Courses_List:
          print(f"{i['ID']:<10}{i['Name']:>40}")
```

```
EE2703                       Applied Programming Lab
EE2003                         Computer Organisation
EE5311                             Digital IC Design
```

Running the list through for loop and storing the data present in the 2 key's of the current dictionary present in loop in 2 variables and printing it with the requried left and right justification as required.

3 Functions for general manipulation

```
[41]: def twosc(x, N=16):
          binary = []
          if (x*1 >= 0):
              for i in range(N):
                  binary.append(x%2)
                  x = x//2
              a = len(binary)
              for j in range(a):
                  print(binary[a - j - 1], end="")
          else:
              x = x*-1
              for i in range(N):
                  binary.append(x%2)
                  x = x//2
              a = len(binary)

              indi=0
              for p in range(a):
                  j=binary[p]
                  if(indi==0):
                      pass
                  else:
                      binary[p]=1-j
                  if(j==1):
                      indi=1
              for j in range(a):
                  print(binary[a - j - 1], end="")

      def twosc_2(x, N=16):
          string = ''
          for i in range(N):
```

```
        string = str(x%2) + string
        x = x//2
    print(string)

print("Method 1")
twosc(10)
print()
twosc(-10)
print()
twosc(-20,8)

print(f"\n")

print("Method 2")
twosc_2(10)
twosc_2(-10)
twosc_2(-20,8)
```

```
Method 1
0000000000001010
1111111111110110
11101100

Method 2
0000000000001010
1111111111110110
11101100
```

METHOD 1:

I have created an array called "binary" which when run through for loop, stores the remainder when the number is divided by "2" until the number becomes "1". But since, in the array the remainder's gets stored in opposite order as required, so while printing the binary number of the given decimal, I am printing the array in "reverse order".

When the given decimal number is "negative", we need to find 2's compliment. But firstly I am following the same series of steps of creating an array with the remainders by turning the given negative number positive. Then running an another for loop which will check "first 1" in the binary array and after this first "1" is noticed, I am overwriting the next subsequent numbers as "1 - number" (that is taking the compliment) and then printing the binary array in reverse order to get 2's compliment of the given negative number.

METHOD 2:

This is a much simple method as compared to the above one. Here I am creating a string which stores the (Number % 2) until the number becomes 1 in opposite order. In case of negative number while finding the modulo negative sign is neglected and then same process is followed giving us the 2's compliment of the output after printing string 's'.

4 List comprehensions and decorators

[21]:
```
[x*x for x in range(10) if x%2 == 0]
```

[21]:
```
[0, 4, 16, 36, 64]
```

For loop for x runs from 0 to 9 times, where "x*x"is printed only when the condition of x%2 = 0 is satisfied and is added to the output list.

[6]:
```
matrix = [[1,2,3], [4,5,6], [7,8,9]]
[v for row in matrix for v in row]
```

[6]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

A for loop is running over all the rows of matrix and then another for loop is running inside it through all the elements present in that row to print the respective element. Thus printing all the elements which are present in the (3*3) matrix in form of a list.

[7]:
```
def is_prime(x):
    if x > 1:
        for i in range(2, int(x**0.5)+1):
            if (x % i) == 0:
                return False
        return True
    else:
        return False

print([x for x in range(100) if is_prime(x) == True], end=" ")
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73,
79, 83, 89, 97]
```

A function is_prime is defined which takes an input argument x and checks whether the number is prime or not.

It directly returns False if x is either a negative number, 0 or 1.

Apart from those numbers, it iterates a for loop in range (2, square root (x)) and if number x is divisible by any one of them I am returning False and if it passes through the for loop, it means it has no divisors and thus I am returning True.

A one liner statement is used to print all the prime numbers from 1 to 100, wherein the condition to get printed is `is_prime(x)` should return true for the particular x.

[14]:
```
def f1(x):
    return "happy " + x
def f2(f):
    def wrapper(*args, **kwargs):
        return "Hello " + f(*args, **kwargs) + " world"
    return wrapper
f3 = f2(f1)
```

```
print(f3("flappy"))
```

```
Hello happy flappy world
```

Two functions are defined here, `f1` and `f2`. Here `f1` takes a single input argument `x` and returns a string `happy + x`. On the other hand `f2` also takes a single input argument and returns up a new function called `wrapper`.

Wrapper takes any number of position arguments (*args) and keyworded arguments (**kwargs), wherein wrapper returns a string by combining `Hello`, `result of function f with given arguments` (f is the input given to f2) and `world`.

Finally we define `f3 = f2(f1)`. So when we call `f3` with input as `flappy`, since `f2` takes in `f1` as argument so `flappy` goes in input to `f1`, thereby it's output is given as input to `f2` after which `wrapper` function is called which as earlier told combines `Hello` + `result of f1` since `f1` is passed to `f2` + `world`.

Thus we get the output as `Hello happy flappy world`.

[16]:
```
@f2
def f4(x):
    return "nappy " + x

print(f4("flappy"))
```

```
Hello nappy flappy world
```

Here a decorator `@f2` is defined above another function `f4`. `f4` takes in only a single argument and returns a string `nappy + x`.

When we call `f4` with input `flappy`, it returns `nappy flappy` and this is passed in `f2` since `f2` is defined as a decorator here, which takes this as input, pass it through the wrapper function, thus getting output as `Hello nappy flappy world`.

5 File IO

[19]:
```
def write_primes(N, filename):
    for i in range(1, N+1):
        if (is_prime(i) == True):
            filename.write(f"{str(i)} ")
            print(f"{str(i)} ", end="")

file = open("apl_1.txt",'w')
write_primes(152, file)
file.close()
```

```
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101 103
107 109 113 127 131 137 139 149 151
```

A function `write_primes` is defined which takes 2 inputs, `N` and `filename`.

It uses `is_prime` function over all integers from `(1, N)` to check whether it is prime or not and if it is prime, it is written inside the file given in the input filename.

Writing in a file is done by the function `file.write()` and subsequent file functions are used to open and close it after use.

6 Exceptions

[6]:
```python
def check_prime(x):
    try:
        x = int(x)
        print(x)
        if(is_prime(x) == True):
            print("Given number is a prime number")
        else:
            print("Given number is not a prime number")
    except ValueError:  #ValueError is a type of Error
        print("Oops! That was no valid number. Try again...")

x = input("Please enter a number: ")
check_prime(x)
```

```
Please enter a number: 352.23
Oops! That was no valid number. Try again...
```

A `check_prime` function is created which takes in a single input `x`.

In `try and expect`, if x is int convertible we check whether the integer x is prime or not using `is_prime` function and printing the statement of what it is.

When x can't be converted to integer, it means the given input is not an integer and so we use `exception handling` to print an error message, otherwise we run normally and check whether it is prime or not.