EE2703 - Week 1

G.Sai Krishna Reddy <ee21b049@iitm.ac.in>

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.*

I opened Notebook Metadata cell and changed the authors name to my name.

Process of running code

This notebook is written in Jupyter and run the cells in the given sequence to generate desired output. This notebook can also be opened in other platforms like: 1. VScode 2. Google colab

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

[1]:
```
print(12 / 5)
```

```
2.4
```

It prints **2.4** because / means simple division and it returns float value.

[1]:
```
print(12 // 5)
```

```
2
```

It prints **2** because // means floor division i.e., it returns floor value of the division.

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

```
10 10 1.0
```

By default it prints space if there is a comma in print statement.As mentioned above it outputs 1.0(float value) for a/b.

2.2 Strings and related operations

[5]:
```
a = "Hello "
print(a)
```

```
Hello
```

Here we assigned Hello to a, so it prints Hello .

[6]:
```
print(a+b)  # Output should contain "Hello 10"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[6], line 1
----> 1 print(a+b)  # Output should contain "Hello 10"

TypeError: can only concatenate str (not "int") to str
```

Its showing error because a is of str datatype and b is of int datatype and we can't add two different data types. To overcome this we have to typecast b to string datatype i.e., **b=str(10)**.

CORRECT CODE

[7]:
```
print(a+str(b))
```

```
Hello 10
```

[ ]:
```
# Print out a line of 40 '-' signs (to look like one long line)
# Then print the number 42 so that it is right justified to the end of
# the above line
# Then print one more line of length 40, but with the pattern '*-*-*-'
```

[1]:
```
print(40*"-")
print(f"{42:>40}")
print(20*"*-")
```

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

Here f allocates 40 spaces to 42 and since its given its right justified,so it prints value from right end.

[8]:
```
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
```

f in the print statement is format string, it is used to insert the recently updated values of the variable inside the **{}**(curly braces) and {b:>10} is used to right justify b to 10 spaces.

[ ]:
```
# Create a list of dictionaries where each entry in the list has two keys:
# - id: this will be the ID number of a course, for example 'EE2703'
# - name: this will be the name, for example 'Applied Programming Lab'
# Add 3 entries:
# EE2703 -> Applied Programming Lab
# EE2003 -> Computer Organization
# EE5311 -> Digital IC Design
# Then print out the entries in a neatly formatted table where the
# ID number is left justified
# to 10 spaces and the name is right justified to 40 spaces.
# That is it should look like:

# EE2703                      Applied Programming Lab
# EE2003                        Computer Organization
# EE5131                            Digital IC Design
```

In the below code format string allocates first 10 spaces to id dictionary and last 40 spaces to name dictionary.I iterated through every dictionary in the list and printed the course id and name following the given condition.

[11]:
```
list=[{"id":"EE2004","name":"Digital Signal Processing"},{"id":"EE2703","name":
 ↪" Applied Programming Lab"},{"id":"EE3001","name":"Solid State Devices"}]

for i in list:
    print(f"{i['id']:<10} {i['name']:>40}")
```

```
EE2004                      Digital Signal Processing
EE2703                        Applied Programming Lab
EE3001                            Solid State Devices
```

3 Functions for general manipulation

[15]:
```
# Write a function with name 'twosc' that will take a single integer
# as input, and print out the binary representation of the number
# as output.  The function should take one other optional parameter N
# which represents the number of bits.  The final result should always
# contain N characters as output (either 0 or 1) and should use
# two's complement to represent the number if it is negative.
# Examples:
# twosc(10): 0000000000001010
# twosc(-10): 1111111111110110
```

```
# twosc(-20, 8): 11101100
#
# Use only functions from the Python standard library to do this.
def twosc(x, N=16):
    pass
```

[1]:
```
def twosc(n,N):
    if n<0:
        n=2**N+n
    bin=""
    while N>0:
        if n%2==1:
            bin="1"+bin
        else:
            bin="0"+bin
        n=n//2
        N=N-1
    return bin
n=int(input())
print(twosc(n,16))
```

```
5
```

```
0000000000000101
```

If the number is **negative**, two's complement of that can be found by binary representation of that **number + 2 power no.of bits**.Then I initialised an empty string, then for binary representation we need remainder with 2 every time we divide the number with 2 and i added the remainder to the string to get binary representation of the number.

4 List comprehensions and decorators

[2]:
```
# Explain the output you see below
[x*x for x in range(10) if x%2 == 0]
```

[2]:
```
[0, 4, 16, 36, 64]
```

[ ]:
```
It calculates squares of even numbers below 10 and makes a list of them.
```

[7]:
```
# Explain the output you see below
matrix = [[1,2,3], [4,5,6], [7,8,9]]
[v for row in matrix for v in row]
```

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

[ ]:
```
It goes from row to row accessing every element in that row and creating a new␣
↪list v.
```

[ ]:
```
# Define a function `is_prime(x)` that will return True if a number
# is prime, or False otherwise.
# Use it to write a one-line statement that will print all
# prime numbers between 1 and 100
```

[14]:
```
def is_prime(x):
    if x<=1:
        return False
    for i in range(2,x//2+1):
        if x%i==0:
            return False
    return True
print([x for x in range(1,101) if is_prime(x)])
```

```
[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]
```

For this I checked if there is a factor in range (2 to half of that number).If there is then its not a prime number.Then I used the same function to make a list of prime numbers below 100.

[2]:
```
# Explain the output below
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
```

Here 'f1(flappy)' returns "happy flappy".The fuction f2 takes "f' as an input and returns function called"wrapper".For the wrapper function we don't need to mention the number of inputs we are giving as it takes any number of arguments(args) and keywordarguments(kwargs).If there's a dictionary in the input we are giving to the function then it comes under keyword arguments.Here the function 'f2' returns"Hello happy flappy world".

[3]:
```
# Explain the output below
@f2
def f4(x):
    return "nappy " + x

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

```
Hello nappy flappy world
```

@f2 is a decorator and function 'f4' is decorated with 'f2' which means f4=f2(f4).So it returns "Hello napy flappy world".

5 File IO

[ ]:
```
# Write a function to generate prime numbers from 1 to N (input)
# and write them to a file (second argument).  You can reuse the prime
# detection function written earlier.
def write_primes(N, filename):
    pass
```

[4]:
```
def is_prime(x):
    if x<=1:
        return False
    for i in range(2,x//2+1):
        if x%i==0:
            return False
    return True
def write_prime(N,filename):
    file=open(filename,"w")
    for x in range(2,N+1):
        if is_prime(x):
            file.write(str(x)+"\n")
    file.close()
N=int(input())
filename="primes from 1 to N"
write_prime(N,filename)
```

```
20
```

Here I used previous function is_prime(x).since we need the numbers in the file, we need to **write** them to the file and the above code creates a text file named **primes from 1 to N** and open(filename,"w") opens the file and since we need to write those vakues in it we mentioned "w".

6 Exceptions

[ ]:
```
# Write a function that takes in a number as input, and prints out
# whether it is a prime or not.  If the input is not an integer,
# print an appropriate error message.  Use exceptions to detect problems.
def check_prime(x):
    pass
x = input('Enter a number: ')
check_prime(x)
```

[ ]:
```
def check_prime(x):
    try:
        x=int(x)
        if x<=1:
          print("The given number is not a prime")
          return
        for i in range(2,x//2+1):
```

```
            if x%i==0:
              print("The given number is not a prime")
              return
          print("The given number is a prime")
      except ValueError as e:
          print("Please enter a valid input(a positive integer)")
x = input('Enter a number: ')
check_prime(x)
```

Here **try and except** statements are used to deal with errors that occur while executing a program.First it runs the code within try statement and if an error occurs and if it matches the error type in except statement then instead of giving an error while compiling, it runs the code within except statement.In the above code if we give input as a string or a negative number or a float number, instead of giving an error it executes except statement.