EE2703 - Week 1

Aayush Patel <ee21b003@smail.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.*

From the Edit Menu, I have changed the notebook metadata to my credentials. `AAYUSH PATEL` `ee21b003@smail.iitm.ac.in`

2 Basic Data Types

2.1 Numerical types

Probelm : Explain the codes given below

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

```
2.4
```

Explanation : This statement performs a *division* of two *integers* and returns a *float* value

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

```
2
```

Explanation : This is a *floor division* which returns an *integer* value, i.e. the quotient of the division.

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

```
10 10 1.0
```

Explanation : This statement prints a, b and a/b where a/b will be a floating point datatype. Print statement has **sep=" "** by default, i.e. all those variables will be printed with a *blank space* between them.

2.2 Strings and related operations

Problem : Explain the code given below

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

```
Hello
```

Explanation : This changes the variable **a** from 10 (*integer*) to "Hello" (*string*) and prints it.

Problem : Output should contain "Hello 10"

[6]:
```
try:
    print(a+b)  # Output should contain "Hello 10"
except:
    print("Error in the print statement.")
finally:
    print("The correct way to print a+b is")
    print(a,b)
```

```
Error in the print statement.
The correct way to print a+b is
Hello  10
```

Explanation : The statement print(a+b) was **wrong** because we can not add an *integer* and a *string*. To solve this we have two ways : - Convert the integer into a *string* datatype then concatenate both *strings*. - use print(a,b) .This will print the *integer* and the *string* with a blank space because the parameter sep of print statement has " " by default (i.e. sep=" ")

Problem : 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 '*-*-*-'

[7]:
```
line1="-"*40          #String Multiplication
line2=" "*40 + "42"
line3="*-"*20
print(line1,line2,line3,sep="\n")
```

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

Explanation : This is the concept of *string multiplication* where a string is copied and concatenated the number of times we want. This output can also have been obtained using loops annd printing the characters one by one but String Multiplication makes this process simpler.

Problem : Explain the given code

[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
```

Explanation : This is an example of **F-strings** in python. F-strings provide a concise and convenient way to embed python expressions inside string literals for formatting. F-strings make *string interpolation* simpler.

Problem : Create a list of dictionaries.

In the dictionaries 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.

[9]:
```
courses_list=[
    {
        "id"   : "EE2703",
        "name" : "Applied Programming Lab"
    },
    {
        "id"   : "EE2003",
        "name" : "Computer Organization"
    },
    {
        "id"   : "EE5131",
        "name" : "Digital IC Design"
    }
]

for course_dict in courses_list:
    id=course_dict["id"]
    name=course_dict["name"]
    print(f"{id : <10}{name : >40}")
```

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

Explanation : The list - *courses_list* contains dictionaries which store the *id* and *name* of courses which are printed here using for loop and F-strings.

3 Functions for general manipulation

Probelm : 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.

[10]:
```
def twosc(x, N=16):
    #Firstly check if x is an integer.
    try:
        x=int(x)
        bin_string=bin(x)
        negative=False
        #Process the bin_string to remove the intial "0b" or "-0b"
        if(bin_string[0]=='-'):
            bin_string=bin_string[1:]
            negative=True
        bin_val=bin_string[2:]
        #Get the required number of bits N
        if(len(bin_val)>N):
            processed_string=bin_val[-N:]
        else:
            l=N-len(bin_val)
            processed_string="0"*l+bin_val
        #Now deal with it if x is negative
        if(negative==True):
            #Take 2's Comliment
            ones_compliment=""
            for i in range(len(processed_string)):
                if(processed_string[i]=='0'):
                    ones_compliment+="1"
                else:
                    ones_compliment+="0"
            carry=1
            twos_compliment=""
            for i in range(len(ones_compliment)-1,-1,-1):
                if(carry==1):
                    if(ones_compliment[i]=='0'):
                        twos_compliment='1'+twos_compliment
                        carry=0
                    else:
                        twos_compliment='0'+twos_compliment
                        carry=1
                else:
                    twos_compliment=ones_compliment[i]+twos_compliment
            print(twos_compliment)
        else:
            print(processed_string)
    except:
```

```
        print(f"{x} is not a integer.")
twosc(10)
twosc(-10)
twosc(-20,8)
```

```
0000000000001010
1111111111110110
11101100
```

Explanation : After confirming that the object **x** which is passed as an argument in the function `twosc()` is indeed an *integer*, I used the inbuilt **bin()** function of the *Python Standard Library*. This returns a string - *bin_string* but this string has prefix **'0b'** for positive numbers and **'-0b'** for negative numbers, which have to be removed. Now add extra 0s in the beginning if **N** is greater than the length of this string.

Print this string if **x** was positive. Otherwise firstly take *One's Compliment* by replacing 0s with 1s and 1s with 0s. Then Take *Two's Compliment* by adding 1 to the number obtained. This can be done bitwise by checking :

- **0 + 1 = 1 (carry = 0)**
- **1 + 1 = 0 (carry = 1)**
- **1 + 1 + 1 = 1 (carry = 1)**.

4 List comprehensions and decorators

Problem : Explain the given code

[11]:
```
# Explain the output you see below
[x*x for x in range(10) if x%2 == 0]
```

[11]:
```
[0, 4, 16, 36, 64]
```

Explanation : This statement generates a *list*.

Then for statement iterates through all the numbers from *x = 0 to 9*, and the if statement checks whether **x** is even. If **x** is even then **x** times **x** is appened to the list. This method of generating list is called `List Comphrension` and this is used to shorten the code a bit for better aesthetics.

Problem : Explain the given code

[12]:
```
# 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]
```

[12]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Explanation : This statement runs a *nested loop*, i.e. a loop inside another loop. It iterates through matrix which is a two dimensional list and returns a list *row*. Now the second loop iterates through the elements of the list *row* and appends them to a *list*.

4.1 Prime Numbers

Problem : 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.

[13]:
```
def is_prime(x):
    if(x<2):
        return False
    i=2
    while(True):
        if(i*i>x):
            break
        if(x%i==0):
            return False
        i+=1
    return True

for i in range(1,101):
    if(is_prime(i)):
        print(i,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
```

Explanation : The function `is_prime()` iterates through all the numbers from 2 to *square root* of n to check if any pair of integers other than 1 and n exists that divides n. And I am calling that function for all number from 1 to 100 to check if they are prime or not.

4.2 Function as Argument

Problem : Explain the given code

[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
```

Explanation : After defining the fucntions **f1()** and **f2()**, a ***wrapper function*** **f3()** is defined as **f2(f1)**. So the argument passed inside **f3()** will first go in **f1()** and the returned object will go as argument in the function **f2()**. Here "flappy" is passed inside f1() to return "happy flappy". Now this is passed inside f2(). *args is used to take arguments of variable length and *kwargs is used to take key-value arguments of variable length.

Problem : Explain the given code

[15]:
```
@f2
def f4(x):
    return "nappy " + x

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

```
Hello nappy flappy world
```

Explanation : This is a simpler way to implement the same thing as above, by wrapping one function inside the other, using a decorator.Decorators allow us to wrap another function in order to extend the behavior of the wrapped function, without permanently modifying it.

5 File IO

Probelm : 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.

[16]:
```
def write_primes(N, filename):
    f=open(filename,"w")
    for i in range(1,N+1):
        if(is_prime(i)):
            f.write(str(i)+" ")
    f.close()
    print("Successfully written to the file.")
write_primes(100,"prime.txt")
```

```
Successfully written to the file.
```

Explanation : Firstly we create a *file object* (file handle) **f**, which opens the file ***prime.txt*** in **write mode**. If this file does not exists then it creates a new file with that name in write mode. Then it iterates through all number from **1 to N** and prints them to a file if it a prime. The statement *f.close()* closes the file and writes to the file if there is any data in the *buffer* of the file handle.

6 Exceptions

Problem : 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.

[17]:
```
def check_prime(x):
    try:
        x=int(x)
        if(x<0):
            raise Exception
        if(is_prime(x)):
            print(f"{x} is prime.")
        else:
```

```
            print(f"{x} is not prime.")
    except:
        print(f"{x} is not a positive integer")
x = input('Enter a number: ')
check_prime(x)
```

```
5 is prime.
```

Explanation : x is taken input as a string. In the function `check_prime()` ,the type conversion to int raises an error if x was a *non integer* object like *float* or *double*. If x is smaller than 0 then also the error is raised. If x is a positive integer then it checks if it is prime or not and prints the corresponding statements.