EE2703 - Week 1

Shah Jainam EE21B122

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.

To edit the author name I have edited notebook meta data from Edit window, and inside authors column changing the name from Nitin Chandrachoodan to Shah Jainam EE21B122

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
```

The operation '/' gives 12 divided by 5 i.e 2.4, and print() displays that to user screen.

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

```
2
```

The operation '//' gives floor value of 12 divided by 5 i.e [2.4] = 2, and print() displays that to user screen.

[3]:
```
a=b=10
print(a,b,a/b)
```

```
10 10 1.0
```

- The first line intializez the variable a and b to be equal and their value being 10
- The second line prints value of a which is 10, b which is also 10 and a/b which is 1.0 as a/b will output floating point value in computer

`In python we need not to specificy the type of variable we need otherwise in programming language like C`

2.2 Strings and related operations

[4]:
```
a = "Hello "
print(a)
```

```
Hello
```

- The first line stores the string Hello in variable a
- The second line prints the vaue stored in a i.e Hello

[5]:
```
print(a+ str(b))  # Output should contain "Hello 10"
```

```
Hello 10
```

The original code was print(a+b) which is not valid as a is string type but b is int type so I typecasted b to string type using str(b)

String concating that is using + to concat two strings only works when all involved variables or data are str type only

[6]:
```
# 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 '*-*-*-'
for i in range(81):
    if i<40:
        print('_',end='')
    elif i==40:
        print('42')
    elif i>40 and i%2==0:
        print('*',end='')
    else:
        print('-',end='')
```

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

- To print out a line of length 40 I used for loop and if statment so I get a line length 40 and overwrite the end in the print function from defualt '/n' to '' so the charcter '_' does'nt get printed in next line

- to print 42 at the end of above line i used elif stament to print 42 when the program is at 40 step i.e after the line of length 40 is printed its right, note here I havent change the end in print('42') because I want to print in new line in third part of the problem
- third elif statement is used to print * on even positions and - on odd positions, with modifications on print function to make line

more info on print syntax -> https://docs.python.org/3/library/functions.html#print

[7]:
```
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
```

The f before object in print means treat that object as formated string literals i.e treat {} in special way instead of printing them as it is

To treat a string quiet literally instead of formating it use 'r' instead of 'f', these are called raw strings literals

[1]:
```
# 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

l = [{'id':'EE2703','name':'Applied Programming Lab'},{'id':'EE2003','name':
 ↪'Computer Organization'},{'id':'EE5131','name':'Digital IC Design'}]

for dic in l:
    print(dic['id'].ljust(10,' '),dic['name'].rjust(40, ' '))
```

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

- First I made a list of dictionary with id and names as keys and their values
- Next I iterated through each dictionary and printed their keys and values and used ljust and rjust to print the context in desired output

3 Functions for general manipulation

[1]:
```
# 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):
    if x>=0:
        x_ = format(x, 'b')
        y = len(format(x, 'b'))
        if N<y:
            return x_[-N:]
        else:
            return format(x, 'b').zfill(N)
    else:
        x = -x
        x_ = format(x, 'b')
        y = len(format(x, 'b'))
        if N<=y:
            inv = ''
            bits = format(x, 'b')
            for i in bits:
                if i == '0':
                    inv = inv + '1'
                else:
                    inv = inv +'0'
            inv = '1'+inv           # I have added this here because in edge␣
 ↪case of -15(01), its only 1 digit(1 no preceding zeroes) to padd the number␣
 ↪with N digits
            ans = bin(int(inv,base=2)+ int('1',base=2))[2:]
            return ans[-N:]
        else:
            inv = ''
            bits = format(x, 'b').zfill(N)
            for i in bits:
                if i == '0':
                    inv = inv + '1'
                else:
                    inv = inv +'0'
```

```
            return bin(int(inv,base=2)+ int('1',base=2))[2:]


print(twosc(-15,3))
#%timeit twosc(-1234567890,1000000)
```

001

- First if the input number x is positive then I have used the inbuilt function bin or format to convert decimal to binary and remove prefix 0b from it, then using inbuilt function zfill I have added the required number of zeroes

for more info on bin https://docs.python.org/3/library/functions.html#bin

- Secondly if x is a negative integer, then we need its 2's complement,for that first I find its ones complement first by inverting 0 to 1 and 1 to 0 using a for loop, note that bin or format returns a string not a int type so taking its advantage I use string concatination to get one's complement
- Now that I have one's complement; binary addition of the 1 to the number gives me two's complement, so for that I have done simple addition of the one's complement and '1' in base 2 note that defailt int base is 10, to change it we have to overwrite it by 2 using the above syntax and also that the number to be changed should be of str type not a int type for the opertaion to work

for more info on int https://docs.python.org/3/library/functions.html#int

- To tackel the edge case of number of digit being less than total number of digits in binary representation (i.e twosc(4,1) should dispay one digit), I check for the condition using if else loop and then use string slicing methods

4 List comprehensions and decorators

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

[10]:
```
[0, 4, 16, 36, 64]
```

This returns a list of the square of all even numbers between 0 and 9

[11]:
```
# 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]
```

[11]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

v for row in matrix takes the first element in the list called matrix and then for v in row takes the first element in list called row and makes a new list of v, i.e a list of all elements in the nested list in the matrix

[3]:
```
# 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

def is_prime(x):
    cat = 0
    #-----------------------------------------
    if x==2 or x==3:
        return True
    elif x==1 or x==0:
        return False
    else:
        for i in range(2,x-1):
            if x%i == 0:
                cat = cat + 1
    #-----------------------------------------
    if cat != 0:
        return False
    else:
        return True

def prime_in_100():
    ls = []
    for i in range(101):
        if is_prime(i):
            ls.append(i)
    return ls


#is_prime(9)
prime_in_100()
```

[3]:
```
[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]
```

[61]:
```
# 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
```

- The f3('flappy') means we are giving function f3 an argument string 'flappy', now f3 is made of f2 and f1, going by the inner most bracket we can see f1 will be executed first and f1 will return 'happy flappy' as it is just concating two strings now this is passed to f2 as a argument which in turn concats 'Hello' and ' world' to the argument and returns it.
- Note that we had to use wrapper function to concat the strings as we have defined f3 as f2(f1) and f1 is a function not a string, and only strings can be concated in python.

[62]:
```
# Explain the output below
@f2
def f4(x):
    return "nappy " + x

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

```
Hello nappy flappy world
```

@f2 is decorator and this is used to do the exact same thing the above code f3=f2(f1) is suppose to do here it will be f2(f4)

5 File IO

[6]:
```
# 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):
    ls =[]
    for i in range(0,N):
        if is_prime(i):
            ls.append(i)
            f = open(filename, "w")
            f.write(str(ls))
            f.close()
write_primes(100, 'raw.py')
```

6 Exceptions

[2]:
```
# 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):
    try:
        x=int(x)
    except ValueError:
        return 'Error provide a integer value'
    #----------------------------------------
    if x==2 or x==3:
            return True
    else:
        cat = 0
        for i in range(2,x-1):
            if x%i == 0:
                cat += 1
    #----------------------------------------
    if cat!=0:
        return False
    else:
        return True


x = input('Enter a number: ')
check_prime(x)
```

```
Enter a number: 2.5
```

[2]:
```
'Error provide a integer value'
```

I have added a try except block at x = int(x) so whenver there is a ValueError I get an exception, this code block only captures Value Error other Error that may be caused are not handle by this, to handle all types of errors; remove ValueError in except statment.