EE2703 - Week 1

Sujal Burad <ee21b144@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.

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)
# This statement prints out the division of two numbers 12, 5. '/' does the␣
 ↪division operation.
```

```
2.4
```

[2]:
```
print(12 // 5)
# This statement prints out the remainder when 12 is divided by 5. '//' is used␣
 ↪to provide the remainder when one number
# is divided by other.
```

```
2
```

[3]:
```
a=b=10
print(a,b,a/b)
# This prints out a, b, and when a is divided by b.
```

```
10 10 1.0
```

2.2 Strings and related operations

[4]:
```
a = "Hello "
print(a)
# It prints out the string a.
```

```
Hello
```

[5]:
```
print(a+b)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [5], in <cell line: 1>()
----> 1 print(a+b)

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

print(a+b)
Output should contain “Hello 10” The compilation of the above statement is done properly, but it gives RunTime, more specifically TypeError.
Since, a is string and b is integer.

[6]:
```
print(a+str(b))
# We have to convert b to string to perform the operation.
```

```
Hello 10
```

[7]:
```
# Print out a line of 40 '-' signs (to look like one long line)

print('-'*40, end="")

# Then print the number 42 so that it is right justified to the end of
# the above line

print("42".rjust(87))
# .rjust() is used to right justify the string with given number of spaces.

# Then print one more line of length 40, but with the pattern '*-*-*-'

print("*-"*20, end="")
```

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

[8]:
```
print(f"The variable 'a' has the value {a} and 'b' has the value {b:>10}")
# {:> } also does the right justification of string.
```

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

[9]:
```
# 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

# Creating a list of dictionaries:
dict=[{'id':'EE2703', 'name':'Applied Programming Lab'},
      {'id':'EE2003', 'name':'Computer Organization'},
      {'id':'EE5311', 'name':'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

# Printing out the entries as per the condition
for i in dict:
    print (i['id'].ljust(10), i['name'].rjust(40))
```

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

3 Functions for general manipulation

3.1 Method 1

The below code is used to find the binary representation of number.
1] We continue dividing the number by 2, till it is greater than zero.
2] Simultaneously the remainder obtained on dividing by 2 is appended to the list.
3] To get the final representation of the number, we reverse the elements of the list.

We continuously divide by 2, because it gives us the number of division steps (n) required to reach the final stage, after which the division of previously obtained quotient is not possible. This n number of division gives us the number of binary positions for the binary digits (0, 1). The n value also defines the power of 2 to be assigned to the binary digits as per its position in final representaion. The power of 2 extends from 0 to (n-1).

Starting from the right extreme, at the ith position, if its 1, then its value is 1 x 2ˆ(i-1), else if it is 0, then its value is 0x2ˆ(i-1).

3.2 Method 2

We can also use the bin() function, which is in-built in python to get the binary representation of number.
It gives us the string format of binary representaion, with "0b" in its beginning.
I created two cases : when number>0 i.e the binary representation, and other when number<0 i.e twos complement representation.
I have commented these code lines. To use that we can uncomment those lines, and comment the lines for Method 1.

[10]:
```
# 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):
    # Method 1
    # The below loop gets us the binary representation of the magnitude of that␣
 ↪number
    absolute=abs(x)
    digi=[]
    while(absolute>0):
        k=absolute%2
        digi.append(k) # To append an element to the list
        absolute=absolute//2

    digi.reverse() # To reverse the elements of list
    binary="" # Adding the list elements to the binary string, to perform␣
 ↪further operations on it.
    for i in digi:
        binary = binary+str(i)

    # Method 2 consists of the lines containing the bin() function.
    # bin() is in-built python function to get binary representation of a number,

    if(x>=0):

#         binary = bin(x).replace("0b", "") # Method 2 line
```

```
        length = len(binary)
#To print extra zeros, if the optional parameter 'N' representing the number of␣
↪bits, is greater than the length of binary value
        print("0"*(N-length), end="")
        print(binary)

    # This case prints out two's complement for negative numbers,
    if(x<0): # For two's complement representation

#         binary = bin(abs(x)).replace("0b", "")  # Method 2 line

        length = len(binary)
        value = ('0'*(N-length))+(binary)
        ones_comp=""

        # The below for loop gives us the ones complement representation
        for i in range(len(value)):
            if(value[i]=='1'):
                ones_comp=ones_comp+'0'
            elif (value[i]=='0'):
                ones_comp=ones_comp+'1'
        print(bin(int(ones_comp, 2)+1).replace("0b", "")) # This gets us the␣
↪twos complement representation of the number
        # int(ones_comp, 2) gets the integer value of ones_comp using base as 2

twosc(10)
twosc(-10)
twosc(-20, 8)
```

```
0000000000001010
1111111111110110
11101100
```

4 List comprehensions and decorators

The below code uses list comprehension:

It gets us the square of number from 0 to 9(10-1), if its even.

The numbers are automatically appended to the end of list.

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

# Square of number from 0 to 9(=10-1) is printed as a list if it is even
```

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

The below code prints out the individual elements of the matrix, one by one, when going through each row.

```
[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]

      # this prints every element in a list, list-by-list
```

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

The below code prints the prime numbers between 1 and 100:

For a number to be prime it should have only two factors, one is 1, and the other the number itself.

Numbers 1 and less than 1 are not considered as prime.

```
[13]: # 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):
          # Number is not prime when it is 1 or less than 1.
          if(x<=1):
              return False
          count=0
          # The below for loop checks that the total number of factors is 2.
          for i in range(1, int(x+1)):
              if(x%i == 0):
                  count=count+1
          if(count==2):
              return True
          else:
              return False

      [p for p in range(1, 100) if(is_prime(p))]
```

```
[13]: [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]
```

1] f1(x) takes an input 'x', concatenates it with "happy", and returns it.
2] f2(f) takes the function 'f' and returns the function wrapper.
3] wrapper functions takes variable length of arguements using the *args and ** kwargs. It returns the string "Hello" concantenated with f, which takes variable length of arguements using *args and ** kwargs, and " world".

'args' is used to pass variable length non keyword argument to the function.
The arguments are passed as a tuple and these passed arguments make tuple inside the function with same name as the parameter excluding asterisk .

'** kwargs' is used to pass variable length of keyword arguments to the function. The arguments are passed as a dictionary and these arguments make a dictionary inside function with name same as the parameter excluding double asterisk ** .

[14]:
```
# 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 is assigned the return function of f2(f1), i.e the wrapper, which is formed␣
 ↪when f1 is fed as the arguemnent to f2().
f3 = f2(f1)
print(f3("flappy"))

# "flappy" is passed as string arguement to f3, which then calls f1("flappy"),␣
 ↪and at the end the expression is returned as
# Hello happy flappy world
```

```
Hello happy flappy world
```

@ is a decorator which is used to extend the functionality of an existing function, Using this f4() works similarly as f3()

[15]:
```
# Explain the output below
@f2

def f4(x):
    return "nappy " + x

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

```
Hello nappy flappy world
```

5 File IO

[16]:
```
# 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.

file = open("prime_nos.txt", "w") # Creating a text file 'prime_nos' and writing␣
 ↪in it
def write_primes(N, filename):
    for p in range(2, N):
        if(is_prime(p)):
            file.write(str(p)+" ")

write_primes(100, file)

# To read the contents of the file, we can uncomment these below lines.
f = open("prime_nos.txt", 'r')
content = f. read()
print(content)
f. close()
```

6 Exceptions

[20]:
```
# 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:
        if(int(x)): # To check if the number is an integer.
            if(is_prime(int(x))):
                print("It is a prime number")
            else:
                print("It is NOT a prime number")
    except ValueError:
        print("The input is not a valid integer")
x = input('Enter a number: ')
check_prime(x)
```

```
Enter a number: 48979853
It is NOT a prime number
```

```
[ ]:
```