EE2703 - Week 1

Varsha S P, EE21B154 <ee21b154@smail.iitm.ac.in>

February 4, 2023

1 Document metadata

This can be done by editing Notebook metadata in Property Inspector.

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

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

```
2.4
<class 'float'>
```

2.2 Explanation

A single slash to divide 2 integers or floating numbers results in type float

[17]:
```
print(12 // 5)
print(type(12 // 5))
```

```
2
<class 'int'>
```

2.3 Explanation

A double slash gives an approximation(step of) of the dividing integers. The type of the resulting value, depends on the input. If one of any two integers are of float type, the resulting type will also be float

[18]:
```
a=b=10
print(type(a))
print(a,b,a/b)
```

```
<class 'int'>
10 10 1.0
```

2.4 Explanation

Two varialbles are assigned interger values and single slash is used to divide them. Thus the resulting value is of type float.

2.5 Strings and related operations

[25]:
```
a = "Hello "
print(a)
```

```
<class 'str'>
```

2.6 Explanation

A varialble is being assigned a random variable and is printed.

[27]:
```
print(a+str(b))# Output should contain "Hello 10"
```

```
Hello 10
```

2.7 Explanation

Two values of same type can be concatenated using a '+' sign.

[84]:
```
# 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 '*-*-*-'

print("-"*40,f"{42:>0}")
print("*-"*20)
```

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

2.8 Explanation

Strings can also be repeated by just using '*' sign.

[29]:
```
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
```

2.9 Explanation

These kind of strings are called Literal String Interpolation or more commonly as F-strings. It is a much simpler way to use varaibles within the string.

[30]:
```
# 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

courses=[{"EE2703":"Applied Programming Lab"},{"EE2003":"Computer␣
 ↪Organization"},{"EE5131":"Digital IC Design"}]
for x in range(len(courses)):
    for x, y in courses[x].items():
        print(f"{x:<10}",f"{y:>40}")
```

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

2.10 Explanation

A list of dictionaries was assigned to 'courses'. For loop was used to print out all the elements in the dictionaries. Function items was used to pick each list and left shift and right shift keys in each list.

3 Functions for general manipulation

[100]:
```
# 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:
        bin = [int(i) for i in list('{0:0b}'.format(x))]
        list_1=list(map(str, bin))
        stri=""
        res = stri.join(list_1)
        final= res.zfill(N)
        return final
    else:
        bin = [int(i) for i in list('{0:0b}'.format(abs(x)))]
        n=len(bin)
        z=pow(2,n)-abs(x)
        nbin=[int(i) for i in list('{0:0b}'.format(z))]
        list_1=[]
        ones=1
        for i in range(N-len(bin)):
            list_1.append(ones)
        zeroes=0
        for i in range(len(bin)-len(nbin)):
            list_1.append(zeroes)
        for x in nbin :
            list_1.append(x)
        string=list(map(str, list_1))
        stri=""
        res = stri.join(string)#concatenating all the elements
        return(res)

    pass
x=int(input("Enter:"))
print(twosc(x,8))
```

```
Enter: -20

11101100
```

3.1 Explanation

To build a fuction that converts decimal to binary, it can be divided into two, if the entered integer is positive or not. To get binary of a positive decimal, we used format code '{0.b}'. It will be list type. To use zfill() to get zeroes before the binary number, it should be in string format. So, we used join() fucntion for a defined stri. Now, for negative integers we know that 2'complemet of its absolute, which is aslo equal to 2^n - N (n=length of binary form of absolute of given integer, N= absolute of given integer). Same set of funtions were used to get the binary of negative integers.

4 List comprehensions and decorators

[7]:
```
# Explain the output you see below
[x*x for x in range(10) if x%2 == 0]
```

[7]:
```
[0, 4, 16, 36, 64]
```

4.1 Explanation

The code checks for all integers from 0 to 9 and if it's an even number, its square gets appended to the list

[36]:
```
# 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]
```

[36]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

4.2 Explanation

Here, matrix is a 2-D array, it goes through every element and concatenates to a list.

[39]:
```
# 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):

    for i in range(2,int(x//2)+1):
        if x%i==0:
            return False

    return True


pass
print([x for x in range(2,100) if is_prime(x)==True])
```

```
[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]
```

4.3 Explanation

To check if an integer is prime or not, a for loop is run. And for the entered range all prime numbers are printed

[41]:
```
# 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
```

4.4 Explanation

Python wrappers are used. Here, f3 is passed as an argument, and f2 is wrapped around f1. Finally f3 is called and it gives "Hello happy flappy world". It's helps modifying the behavior of functions without changing its original behaviour.

[43]:
```
# Explain the output below
@f2
def f4(x):
    return "nappy " + x

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

```
<class 'str'>
```

4.5 Explanation

Here, f2 wrapps around f4 and prints the string. It generates "nappy flappy" passed into function f2, that generated the string, inside wrapper function, "Hello nappy flappy world".

5 File IO

[74]:
```
# 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):
    file = open(filename,'w')
    for i in range(2,N+1):
        if is_prime(i):
            file.write(str(i)+ " ")

    file.close()

    return
write_primes(100, "prime.txt")
```

5.1 Explanation

It opens a file by the file name prime, and saves the list of prime numbers in the range(1,N).

6 Exceptions

[60]:
```
# 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:
        val = int(x)
        if val<0:
            print("You have not entered a positive integer")
        elif is_prime(val):
            print("Entered number is a prime number")
        else:
            print("Entered number is not a prime number")

    except ValueError:
        print("You have not entered an integer")



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

```
Enter a number:  04
```

```
It is not prime
```

6.1 Explanation

Here, try and exception handeling method is used to find if an entered number is a prime or not. It also checks if it's a positive integer, decimal and displays appropriate message.