EE2703 - Week 1

GVV Praneeth Reddy <EE21B048>

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.

Changing the author name

1. Go to property inspector
2. Go to advanced tools
3. In document meta data section, change the author name and email address which is in JSON format

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).

3 Process of running the code

This file can uploaded to a jupyter server or google collab notebook to run.Please run all the code cells in the sequence to generate correct outputs.

3.1 Numerical types

[1]:
```python
print(12 / 5) #Normal Division
```

```
2.4
```

Normal Division: '/' is the normal division operator in python which gives a float value with upto 15 decimal places.

[2]:
```
print(12 // 5) #Floor Division
```

```
2
```

Floor Division: '//' is the floor division operator in python which gives the greatest integer less than or equal to the normal division value and it will be a int value if both the diviend and divisor are int and a float value if one of them is float.

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

```
10 10 1.0
```

- Here the variables 'a' and 'b' are being declared and initialized to the integer '10' and the 'print()' function prints the value of a, b and a/b(normal division of a and b) with a single space between them

3.2 Strings and related operations

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

```
Hello
```

- Now the variable 'a' is changed to the string 'Hello' from initial integer 10 and print(a) will print the string 'Hello'

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

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[5], line 1
----> 1 print(a+b)  # Output should contain "Hello 10"

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

- The variable a is a string and b is an integer we can only concatenate strings but not a string and an integer

[6]:
```
print(a+str(b))
```

```
Hello 10
```

- The integer b can be typecasted to a string using str(b) so that 10 will be treated as a string and can be concatenated with 'a' and the concatenated strings will be printed with a space between them.

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

[75]:
```python
for i in range(0,40):
    print("-", end="")
# print("\n")
print(f"\n{42:>40}")
for j in range(0,20):
    print("*-", end="")
```

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

- Here a for loop is being used to print 40 '-' signs. The variable i takes the values from 0 to 39 when the for loop runs. The print() function generally prints a new line character at the end but when can change that using end=""".So here end="" helps us to print all the '-' signs the same line. :>40 is used to to right justify it to the end of the first line. The second for loop is used to print 20 '*-' signs.

[8]:
```python
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
```

- Here f in the print statement is used to format the output the {} inside the print statement will print the value of the variable in the {} and everything else will be treated as a normal text and will be printed as it is and and :>10 is used to right justify the value of b by 10 spaces.

[ ]:
```python
# 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
```

[10]:
```python
list = [{"id" : "EE2073", "name" : "Applied Programming Lab"}, {"id": "EE2003",␣
↪"name": "Computer Organization"}, {"id": "EE5311", "name": "Digital IC␣
↪Design"}]
for i in range(3):
    print(f"{list[i]['id']:<10}{list[i]['name']:>40}")
```

```
EE2073                     Applied Programming Lab
EE2003                       Computer Organization
```

- We can define a list using square brackets[] and a dictionary using curly brackets{}. The elements of a list are seperated by ','. Each dictionary contains some key value pairs which are seperated by ',' and the key value pairs are defined in this way key : value.The for loop is used to iterate through all 3 indices of the list and print the values assigned to the keys. :<10 is used to left justify the ID number by 10 spaces and :>40 is used to right justify the course name by 40 spaces

4 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:
        print(bin(x)[2:].zfill(N))
    else:
        print(bin(2**N+x)[2:])
    pass
a=int(input("Enter a Number: "))
twosc(a, N=8)
```

```
Enter a Number:  -20

11101100
```

- Here a function named twosc is defined using def with two parameters x and N. Bin() function converts an integer to binary number and adds 0b at the left end so [2:] is used to exclude that. If we have a non negative number as input we are directly using bin(x) and if it is negative we are adding 2^N to it to get the two's compliment. zfill(N) is used to add zeros to the left of the value produced until it has N characters.

5 List comprehensions and decorators

[22]:
```
# Explain the output you see below
[x*x for x in range(10) if x%2 == 0]
```

[22]:
```
[0, 4, 16, 36, 64]
```

- This list contains the values of x*x for the values of x from 0 to 9 which satisfies the condition x%2==0(remainder of x when divided by 2 is 0).

[25]:
```
# 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]
```

[25]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- Here the variable matrix is a 2D array and this list contains all the elements of the array. Here it goes through a nested for loop.

[ ]:
```
# 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
```

[2]:
```
def is_prime(x):
    if x==0 or x==1:
        return False
    else:
        for i in range(2,int(x**0.5)+1):
            if x%i==0:
                return False
        return True
print([i for i in range(1,101) if is_prime(i)])
```

```
[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]
```

- A function named is_prime is defined which checks if the number is divisble by any number between 2 and square of the number and if it is divisible it says that is_ prime is true else false. Here the list comprehension is used to write a one line statement that print all the prime numbers between 1 and 100. this lists contains all the values of i which are between 1 to 100 and satisfies is_prime(i)==true.

[83]:
```
# 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
```

- A function f1 is defined which takes an argument x and returns a string “happy” concatenated with x. function f2 takes another function f as an argument and returns a new function wrapper. *args and **kwargs allows us to variable number of arguments to the function.variable f3 refers to the function wrapper returned by f2 and it calls the function f1 which takes the value of x as the string “flappy”. so it will print the string “Hello happy flappy world”

[84]:
```
# Explain the output below
@f2
def f4(x):
    return "nappy " + x

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

```
Hello nappy flappy world
```

- @f2 is a decorator, which implies f4=f2(f4). So f2 calls the function f4 first which takes flappy as input and returns Hello nappy flappy world

6 File IO

[3]:
```
# 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(1,N+1):
        if is_prime(i):
            file.write(str(i) + " ")
    file.close()
    pass

n= int(input("Enter a Number"))
write_primes(n, "prime_numbers.txt")
```

```
Enter a Number 100
```

- open() opens a file with the given file name and w tells that we are going to write into the file write() is used to write into the file is_prime is used to check if the given number is a prime number. write_primes is defined to write all prime numbers between 1 to N(which isgiven as input) and the filename is another parameter forthe function.close() will close the file after writing.

7 Exceptions

[4]:
```
# 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)
```

```python
        a=1
        if x==0 or x==1:
            print(f"{x} is not a prime number")
        else:
            for i in range(2,int(x**0.5)+1):
                if x%i==0:
                    a=0
                    break
        if a==1:
            print(f"{x} is a prime number")
        else:
            print(f"{x} is not a prime number")
    except ValueError:
        print("Input must be an integer")
x = input('Enter a number: ')
check_prime(x)
```

```
Enter a number:  20.5
```

```
Input must be an integer
```

- The function check_prime takes an input x which is a string and tries to convert it to integer if it is successful it continues to check if the given number is prime or not and prints that else it will give a ValueError printing the statement ‘Input must be an integer’. we are using the try and except to do this.