Sheikh-Sedat Touray

DSP 562

Spring 2024

Problem Set 1 (10 points)

Python Language Basics

Week 1 References: https://jupyter.uri.edu (https://jupyter.uri.edu) **JupyterLab FAQ**

Get an account on jupyter.uri.edu so you can start running Python right away

Upload this Jupyter Notebook to your jupyter.uri.edu account and start filling in the code below the descriptions.

Week 1 References: McKinney, Chapters 1 - 3

#1 Name 3 essential Python libraries:

NumPy: NumPy contains a fast and efficient multidimensional array object ndarray, facilitates mathematical operations between arrays, tools for reading and writing array-based datasets to disk, and thus provides support for large, multi-dimensional arrays and matrices, along with mathematical functions to operate on these arrays. NumPy is a cornerstone library for data manipulation and analysis in the Python ecosystem.

Pandas: Provides data structures like DataFrame for efficient manipulation and analysis of structured data. Pandas is widely used for tasks such as data cleaning, exploration, and preparation before analysis because it provides convenient indexing abilities which easily allows us to reshape, slice and dice, perform aggregations and select subsets of data.

Matplotlib: Is a 2D plotting library for creating static, animated, and interactive visualizations in Python. It provides a wide variety of plots and charts, making it essential for data visualization tasks. Matplotlib is often used in conjunction with other libraries like NumPy and Pandas.

#2 Use a Python command to Display the words: 'Hello world'

In [78]:
```
#Print hello world
print("Hello world")
```

```
Hello world
```

#3 Show a comment or commented line of code

In [75]:
```
# This is a comment in Pythod
# The hash symbol makes it a comment
#commenting on the code below to print hello world
print("Hello world")
```

```
Hello world
```

#4 Define a function that appends an element to a list - defined by square brackets []

In [64]:
```
#Data list defined by []
data = [5,10,15]
#creating the define function
def append_list(data, element):
    data.append(element)
    return data
```

#5 Run the function, appending an element

In [66]:
```
# when we return and run the function
# nothing happened so now we shal append add a new member to the list

# 20 is the new member we are going to add to list
new = 20

#Append and store it to the result variable
result = append_list(data, new)

#print to see what it look like
print(result)
```

```
[5, 10, 15, 20, 20]
```

#6 Define a variable with a string value. Use a method to capitalize the first letter of the string.

In [8]:
```
# Define a variable with a string value
my_case = "the problem is an opportunity!"

# Use the capitalize() method to capitalize the first letter
caps_case = my_case.capitalize()

# Print the original and capitalized strings
print("Original String:", my_case)
#This adds a space between original and caps strings
print('')
print("Capitalized String:", caps_case)
```

```
Original String: the problem is an opportunity!

Capitalized String: The problem is an opportunity!
```

#7 Write a string that spans multiple lines

In [69]:
```
# Defining the multi_line_string
multi_line_string = """This is a string
that spans multiple lines
using double triple-quotes.
It can contain line breaks
and keeps the formatting."""

#Calling multi line string to show output
multi_line_string
```

Out[69]:
```
'This is a string\nthat spans multiple lines\nusing double triple-quotes.\nIt can contain line breaks\nand keeps the formatting.'
```

In [73]:
```
#printing the several lines of text using the new line symbol \n
print('This is a string.\nIt spans multiple lines.\nUses double triple-quotes.\nIt can contain line breaks.\nAnd keeps the formatting.')
```

```
This is a string.
It spans multiple lines.
Uses double triple-quotes.
It can contain line breaks.
And keeps the formatting.
```

#8 Control flow: Give Examples of if, elif, and else

In [15]:
```
#choose a reference number that the control flow is going to be dependent on

ref_num = 10

if ref_num > 0:
    print("Reference is positive.")
elif ref_num == 0:
    print("Reference is zero.")
else:
    print("Reference is negative.")
```

```
Reference is positive.
```

An if statement can be optionally followed by one or more elif blocks and a catchall else block if all of the conditions are False. If any of the conditions are True, no further elif or else blocks will be reached.

#9 Give an example of a for loop

In [38]:
```
#Define the countries in a list
country = ["USA", "Germany","Gambia", "Spain","Dubai","France", "Italy", "Gabon", "Greece", "Portugal"]

#for loop with if statement to skip countries taht start with G
for name in country:
    if name.startswith('G'):
        #printing skipped countries with results
        print(f"Skipping {name}...")
        continue  # Continue to the next iteration without executing the rest of the loop body
    else:
        print(name)
```

```
USA
Skipping Germany...
Skipping Gambia...
Spain
Dubai
France
Italy
Skipping Gabon...
Skipping Greece...
Portugal
```

In [40]:
```
# Now instead of printing the countries we will just remove them for now
# For better visualization of code results
country = ["USA", "Germany","Gambia", "Spain","Dubai","France", "Italy", "Gabon", "Greece", "Portugal"]

counter = []
for name in country:
    if name.startswith('G'):
        continue  # Continue to the next iteration without executing the rest of the loop body
    else:
        print(name)
```

```
USA
Spain
Dubai
France
Italy
Portugal
```

#10 Give an example of a while loop

In [42]:
```
# Using a while loop to count from 1 to 10, skipping even numbers
number = 1

while number <= 10:
    if number % 2 == 0:
        # Skip even numbers
        number += 1
        continue  # Continue to the next iteration without executing the rest of the loop body

    print(number)
    number += 1
```

```
1
3
5
7
9
```

A while loop specifies a condition and a block of code that is to be executed until the condition evaluates to False or the loop is explicitly ended with break.

#11 Give an example of the 'no-op' statement: pass

In [2]:
```
#no-op statement in a for loop
for i in range(4):
    if i<0:
        print("negative!")
    elif i==0:
        print("neutral!")
        pass #place holder
    else: print("positive!")
```

```
neutral!
positive!
positive!
positive!
```

#12 Give an example of the range function

In [16]:
```
# Example: Using the range function to generate a sequence of numbers
numbers = range(5)  # Generates numbers from 0 to 4

# Accessing elements in the range
print("Generated numbers:", list(numbers))
print('')
# Example: Using the range function with start, stop, and step
custom_range = range(1, 15, 3)  # Generates odd numbers from 1 to 13 with a step of 3

# Accessing elements in the custom range
print("Custom range:", list(custom_range))
print('')
# Example: Using range with the length of a list
my_list = [10, 20, 30, 40, 50]

# Creating a range based on the length of the list
index_range = range(len(my_list))

# Accessing elements using the range
for i in index_range:
    print(f"Element at index {i}: {my_list[i]}")
```

```
Generated numbers: [0, 1, 2, 3, 4]

Custom range: [1, 4, 7, 10, 13]

Element at index 0: 10
Element at index 1: 20
Element at index 2: 30
Element at index 3: 40
Element at index 4: 50
```

#13 Give an example of a ternary expression

In [17]:
```
# Example: Ternary expression to determine if a number is even or odd
number = 8

result = "Even" if number % 2 == 0 else "Odd"

print(f"The number {number} is {result}.")
```

```
The number 8 is Even.
```

checks if the number is even or odd. If the condition number % 2 == 0 is true, it assigns the string "Even" to the variable result; otherwise, it assigns "Odd". The result is then printed.

Ternary expressions are useful for writing compact conditional statements in situations where a full if-else block is not necessary.

#14 Give an example of a Tuple data structure

A tuple is a fixed-length, immutable sequence of Python objects which, once assigned, cannot be changed. The easiest way to create one is with a comma-separated sequence of values wrapped in parentheses

In [19]:
```
# you can leave out parenthesis
tup1= 4,5,6

#make a list,sequence, operator using tuple function
tup2 = tuple([5,2,3,1])

#make a string tuple using tuple function
tup3 = tuple("string")

print('tupf:', tup1)
print("")
print('tups:', tup2)
print("")
print('tupt:', tup3)
print("")


#Elements can be accessed with square brackets [] as with most other sequence types.
tup1[1],tup2[3],tup3[5]

#tuples can be unpacked
#example
a,b,c,d =tup2
print("b = ",b)
print('')
#You can concatenate tuples using the + operator to produce longer tuples.
contup = tup1+tup3
print("concatenate: ",contup)
```

```
tupf: (4, 5, 6)

tups: (5, 2, 3, 1)

tupt: ('s', 't', 'r', 'i', 'n', 'g')

b =  2

concatenate:  (4, 5, 6, 's', 't', 'r', 'i', 'n', 'g')
```

#15 Give an example of a List data structure

In [11]:
```
# Example: List representing a collection of colors
colors = ["red", "white", "blue","white", "green"]

# Accessing elements in the list
print("First color:", colors[0])
print("Second color:", colors[1])
print("All colors:", colors)
```

```
First color: red
Second color: white
All colors: ['red', 'white', 'blue', 'white', 'green']
```

#16 Give an example of a dict data structure

In [10]:
```
# Example: Dictionary representing information about a person
dict_P = {
    "name": "Sedat T",
    "age": 100,
    "city": "Kingston",
    "email": "Sedattt@uri.com"
}

# Accessing values in the dictionary
print("Name:", dict_P["name"])
print("Age:", dict_P["age"])
print("City:", dict_P["city"])
print("Email:", dict_P["email"])
```

```
Name: Sedat T
Age: 100
City: Kingston
Email: Sedattt@uri.com
```

#17 Functions are objects. Add the missing code in the function definition to make it work:

In [54]:
```
states = ["   Alabama ", "Georgia!", "Georgia", "georgia", "FlOrIda",
          "south   carolina##", "West virginia?"]
```

In [55]:
```
#! ipython verbatim id=fb7c1a7bea534d85ba3b735097ad0235
import re
# we add def to make it our custom function
def clean_strings(strings):
    result = [] #empty list to store results
    for value in strings:
        value = value.strip()
        value = re.sub("[!#?]", "", value)
        value = value.title() #correct by adding value
        result.append(value)
    return result
```

In [58]:
```
cleaned_states = clean_strings(states)#replace value with states
#Print out the cleaned states
print(cleaned_states)
```

```
['Alabama', 'Georgia', 'Georgia', 'Georgia', 'Florida', 'South   Carolina', 'West Virginia']
```

#18 Generators: fill in the missing code to make it work: generate squares from 1 to 100

In [43]:
```
# we add def to make it our custom function
def squares(n=10):
    print(f"Generating squares from 1 to {n}")
    for i in range(1, n + 1):
        yield i ** 2
```

In [44]:
```
# Provide the desired value of n
gen = squares(10)
```

In [45]:
```
#create a for loop to iterate
for x in gen:
    print(x, end=" ")
```

```
Generating squares from 1 to 10
1 4 9 16 25 36 49 64 81 100
```

#19 Open a text file for reading or writing using the built-in open function

In [60]:
```
#Define the file path
file_path = "array_ex.txt"


# Open the file for writing (create if it doesn't exist)
with open(file_path, "w") as file:
    file.write("Hello, this is a sample text.")

# Open the file for reading
with open(file_path, "r") as file:
    content = file.read()
    print("File content:", content)
```

```
File content: Hello, this is a sample text.
```

#20 Close the file

In [63]:
```
# Close the file
file.close()
```

Download this *.ipynb* notebook file and submit it to your Brightspace Assignment Problem Set 1.