Python Course

“ A Python Course Notes, Examples, Problem Solving, Advices, and some small projects “

By

Ahmad Jawabreh

What prerequisites should be in place before commencing the course?

1- Code Editor Selection: Numerous options are available, with the most renowned being VS Code, or alternatively, you can opt for the PyCharm IDE.

2- Python Installation and Configuration: Begin by installing and setting up Python from the official website.

3- VS Code Python Extension: If you choose VS Code as your code editor, install the Python extension to enhance your development environment.

4- Prerequisite Programming Knowledge: Ensure you possess basic programming knowledge before diving into the Python course.

5- Command Line Proficiency: Familiarize yourself with command line operations, as it is a beneficial skill for Python development. Some Command Line notes are attached in this repository.

6- Please review the file named "notes.txt" in this repository before commencing the course.

1.0 Print Statement:

The print statement is commonly employed as a useful means to display information on the debugging console. However, it is crucial to emphasize that the use of print statements is restricted when working on production code.

So, how can we print some information on the debugging console?

Example: print(“ Hello World”)
By running the app you’ll be able to see the “ Hello World

2.0 Implicit Line Continuation (Optional Semicolon):

If you’ve noticed in the last example that we are not using semicolon at the end of the line as we used to do, because Python's avoidance of semicolons in its code is a characteristic of a programming language feature known as "optional semicolons" or "implicit line continuation." In Python, the end of a line generally marks the end of a statement, and the use of semicolons to terminate statements is optional. The interpreter relies on indentation to determine the structure of the code, making semicolons unnecessary for line termination in most cases. This approach enhances readability and contributes to Python's clean and concise syntax.

3.0 Comments:

In Python, the hash sign is employed to comment on a specific line. For instance: # This is a single-line comment If you are using VS Code, you can conveniently comment on a specific line using the shortcut CTRL+/.

Note: It's essential to note that Python does not have explicit syntax for multiple-line comments. While some use triple quotes """ HERE """ for this purpose, it's crucial to recognize that this construct is, in fact, a string not assigned to any variable, making it an unconventional method for commenting in Python.

4.0 Data Types:

Python Data Types	
Integer	10 , 100 , -10 , -100
Floating	14.19 , 115.59 , -10.23 , -19.28
String	"Hello", "HI", "1232"
List (Array)	[1,2,3,4], ['A', 'B ', 'C'], [1,-1,'A']
Tuple	(1,2,3,4), ('A', 'B ', 'C'), (1,-1,'A')
Dictionary	{"One: 1", "Two: 2", "Three, 3"}
Boolean	2 = = 2

figure 1: Data Types in Python

- int (Integer): Any positive or negative number without any decimal or fractional part is integer
- float (Floating): Any positive or negative number with any decimal or fractional part is integer
- str (String): String type for representing text, e.g., "hello", 'Python'
- List: Ordered, mutable sequence of elements that can hold any type of data
- Tuple: Ordered, immutable sequence that can hold any type of data
- dict (Dictionary): A collection of key-value pairs
- bool: Boolean type representing True or False

4.1 List VS Tuple:

- List: Lists are mutable, meaning you can modify their elements, add new elements, or remove existing ones after the list is created. Use lists when you have a collection of items that may need to be modified, such as adding or removing elements. Lists are suitable for sequences where the length may change during the program's execution.

- Tuple: Tuples are immutable, and once a tuple is created, you cannot change, add, or remove elements. Use tuples when you want to create an immutable and ordered collection. Tuples are useful for representing fixed collections of items, like coordinates or settings, where you don't want the values to be changed accidentally.

5.0 Variables:

Python uses the concept of dynamic typing or duck type which is used also by other programming languages such as PHP and Dart. In Python, variables are dynamically typed, which means their type is determined at runtime based on the value assigned to them. Unlike statically-typed languages where you explicitly declare the variable type, Python allows more flexibility by inferring the type based on the assigned value. This dynamic typing contributes to the language's flexibility and ease of use.

Example:

Variable assignment in Python (dynamic-typed):
myVariable = “Hello World”

VS

Variable assignment in Dart (statically-typed):
String myVariable = “Hello World”

5.1 Variables Rules:

5.1.1 Variables Naming Rules:

- Variable names can consist of letters (both lowercase and uppercase), digits, and the underscore character _.
- They cannot start with a digit.
- Variable names are case-sensitive, meaning myVar and myvar would be considered different variables.
- Avoid using Python reserved words (keywords) as variable names.

5.1.2 Variables Assignment Rules:

Variables are assigned using the assignment operator =. For example: my_variable = 10.

5.1.3 Data Types:

Python is dynamically typed, so you don't need to explicitly declare the data type of a variable. The interpreter infers the type based on the assigned value.

5.1.4 Convention for Variable Names:

It's a convention in Python to use lowercase names with underscores to represent variables (snake_case). For example: my_variable.

5.1.5 Avoid Single Character Names:

Unless it has a specific purpose (like loop counters), it's generally better to use descriptive names rather than single-character names.

5.1.6 Readability Counts:

Choose variable names that are descriptive and convey the purpose of the variable. This enhances the readability of your code.

Note: Python uses the concept of Variable Reassignment, where variables are mutable, meaning their values can be changed during the execution of the program.

6.0 Escape Character Sequences

- Backspace: using \b we can make a backspace in our string, where the character before the \b will be deleted

  Example: print “Hello \b World” OUTPUT: HelloWorld.

- Escape New Line: so if you want to write a string line in multiple lines in your code editor but you want to show it on the same lien when you print it you can use the backslash sign to do that

  Example: print(“ Hello \
  World\
  Baby”)
  OUTPUT: Hello World Baby

- Escape Backslash: if you want to write a backslash sign in you’re string you just need to put one more backslash before it

  Example: print(“ Hello \\ World”)
  OUTPUT: Hello \ World

- Escape Single and Double Quote Sign: if we want to write a single or double quote sign in our string we can use also the backslash sign

  Example: print(‘ Hello \‘World\’ ’)
  OUTPUT: Hello ‘World’

  Example: print(“ Hello \”World\” “)
  OUTPUT: Hello “World”

- Carriage Return: moves the cursor to the beginning of the line

  Example: print("Hello\rWorld")
  OUTPUT: World

  Example: print("123456\rAbcd")
  OUTPUT: abcd56

7.0 String Concatenation:

String concatenation in Python refers to the process of combining two or more strings into a single string. This can be done using the + operator, which concatenates or joins strings.

Example:
str1 = "Hello"
str2 = "World"
mistake = str1 + str2
result = str1 + " " + str2
print(result)

OUTPUT:
HelloWorld
Hello World

In this example, the strings "Hello" and "World" are concatenated with a space in between, resulting in the output.

7.1 String Concatenation Examples:

Example:
Concatenate the variables with some words to produce the message in the following output

OUTPUT: "Hello, My Name Is Ahmad Jawabreh And I'm 21 Years Old and I Live in Palestine."

Variables:
Name = “Ahmad”
Last Name = “Jawabreh”
Age = 21
Live = Palestine

CODE:

print("Hello, My Name Is " + name + " " + last_name + " And I'm " + age + " Years Old and I Live in " + live)

8.0 Strings in Python

8.1 String Methods:

1-) len() :
returns the length of the string

Example:
```
myString = "I Love You"
print(len(myString))
```

OUTPUT:
10

2-) strip()
[strip, rsrip, lstrip]:

- strip: removes the whitespaces from the beginning and the end of the string, you can also specify any character instead of the whitespaces.

- rstrip: removes the whitespaces from the end (right) of the string, you can also specify any character instead of the whitespaces.

- lstrip: removes the whitespaces from the beginning (left) of the string, you can also specify any character instead of the whitespaces.

Example:
```
myString = "       I Love You          "
print(myString.strip())
```

OUTPUT:
I Love You

Example:
```
myString = "       I Love You"
print(myString.lstrip())
```

OUTPUT:
I Love You

Example:
```
myString = "I Love You     "
print(myString.rstrip())
```

OUTPUT:
I Love You

3-) title(): returns the string as title (First letter of every word is capital letter)
Example:
```
myString = "i love you"
print(myString.title())
```

OUTPUT:
I Love You

4-) capitalize(): capitalizes only the first character of the string.

Example:
```
myString = "i love you"
print(myString.capitalize())
```

OUTPUT:
I love you

5-) zfill(): used to pad a string representation of a number with zeros on the left.

Example:
```
number = "42"
width = 5
result = number.zfill(width)
print(result)
```

OUTPUT:
00042

6-) upper(): used to change the case of the characters to uppercase in a string.

Example:
```
text = "hello world"
result = text.upper()
print(result)
```

OUTPUT:
```
HELLO WORLD
```

7-) lower(): converts all the characters in a string to lowercase.

Example:
```
text = "Hello World"
result = text.lower()
print(result)
```

OUTPUT:
```
hello world
```

8-) split(), rsplit(): used to split a string into a list of substrings based on a specified delimiter.

Example:
```
text = "apple orange banana"
result = text.split()
print(result)
```

OUTPUT: ['apple', 'orange', 'banana']

9-) center(): used to center a string within a specified width by make whitespace before and after the string

Example:
```
text = "Hello World"
result = text.center()
print(result)
```

OUTPUT: Hello World

Example:

```
text = "hello"
width = 10
fillchar = '*'
result = text.center(width, fillchar)
print(result)
```

OUTPUT: **hello***

10-) count(): used to count the occurrences of a specified substring within the given string

Example:

```
text = "hello world hello"
substring = "hello"
result = text.count(substring, 6, 18)
print(result)
```

OUTPUT: 1

11-) swapcase(): used to swap the case of each character in the given string. It converts uppercase characters to lowercase and vice versa.

Example:

```
text = "Hello World"
result = text.swapcase()
print(result)
```

OUTPUT: hELLO wORLD

12-) endswith(): This method returns True if the string ends with the specified suffix; otherwise, it returns False.

string.endswith(prefix, start, end)

Example:

```
text = "Hello World"
result = text.endswith("World")
print(result)
```

OUTPUT: True

13-) startswith(): This method returns True if the string starts with the specified prefix; otherwise, it returns False.
string.startswith(prefix, start, end)
Example:
```
text = "Hello World"
result = text.startswith("Hello")
print(result)
```

OUTPUT: True

14-) index(): used to find the index of the first occurrence of a specified substring within the given string. If the substring is not found, it raises a ValueError.
string.index(substring, start, end)
Example:
```
text = "Hello World"
substring = "lo"
result = text.index(substring)
print(result)
```

OUTPUT: 3

15-) find(): used to find the index of the first occurrence of a specified substring within the given string. If the substring is not found, it returns -1.
string.find(substring, start, end)
Example:
```
text = "Hello World"
substring = "lo"
result = text.find(substring)
print(result)
```

OUTPUT: 3

16-) rjust(), ljust(): rjust() and ljust() are string methods used for right-justifying and left-justifying a string within a specified width, respectively. These methods are often used for formatting text in a fixed-width field.
string.rjust(width, fillchar)
string.ljust(width, fillchar)

Example:
```
text = "hello"
width = 10
fillchar = '*'
result = text.ljust(width, fillchar)
print(result)
```

OUTPUT: hello*****

17-) splitlines(): used to split a multi-line string into a list of lines. It identifies line breaks in the string and separates the text accordingly.
string.splitlines(keepends)
Example:
```
text = "Hello\nWorld\nPython"
result = text.splitlines()
print(result)
```

OUTPUT: ['Hello\n', 'World\n', 'Python']

18-) expandtabs(): used to replace tab characters (`\t`) in a string with a specified number of spaces.
string.expandtabs(tabsize)
Example:
```
text = "Hello\tWorld\tPython"
result = text.expandtabs(tabsize=4)
print(result)
```

OUTPUT: Hello World Python

19-) istitle(): checks whether each word in a string starts with an uppercase letter and the rest of the characters are lowercase. It returns True if the string is in title case format; otherwise, it returns False.
string.istitle()
Example:
text = "Hello\tWorld\tPython"
result = text.expandtabs(tabsize=4)
print(result)

OUTPUT: True

20-) isspace(): used to check if a string consists only of whitespace characters. It returns True if all characters in the string are whitespace (spaces, tabs, and newlines), and False otherwise.
string.isspace()
Example:
text = " \t\n"
result = text.isspace()
print(result)

OUTPUT: True

21-) islower(): used to check if all alphabetic characters in a string are lowercase
string.islower()
Example:
text = "Hello World"
result = text.islower()
print(result)
OUTPUT: False

22-) isupper(): used to check if all alphabetic characters in a string are lowercase
string.isupper()
Example:
text = "Hello World"
result = text.isupper()
print(result)

OUTPUT: True

23-) isidentifier(): used to check if a given string is a valid identifier. An identifier is a name given to entities like variables, functions, classes, etc., in a program. For a string to be considered a valid identifier.
string.isidentifier()
Example:

```
text = "variable_name"
result = text.isidentifier()
print(result)
```

OUTPUT: True

24-) isalpha(): used to check if all characters in a given string are alphabetic.
string.isalpha()
Example:

```
text = "Hello123"
result = text.isalpha()
print(result)
```

OUTPUT: True

25-) isalnum(): used to check if all characters in a given string are alphanumeric, It returns True if all characters are either letters (alphabetic) or numbers (digits), and False otherwise.
string.isalnum(),
Example:

```
text = "Hello123"
result = text.isalnum()
print(result)
```

OUTPUT: True

8.1 Strings Examples:

Example:
Create three variables: your name, age, and country, then print the following message and Concatenate the variables with the following tags and words to get the same message in the end. You must show the message as it is with all the tags in the same order and spacing.

OUTPUT: "Hello 'Ahmad', How Are You Doing \ """ Your Age Is "21"" + And Your Country Is: Palestine

CODE:

```
name = "Ahmad"
age = "21"
live = "Palestine"

print(" \"Hello \"'+name+"\', How Are You Doing \\ \"\"\" Your Age Is \"" +age+ "\"\" + And Your Country Is: "+live)
```

Example:
Print the same message as before, but on more than one line. See the desired output

OUTPUT:

```
"Hello 'Osama', How You Doing \
""" Your Age Is "38"" +
And Your Country Is: Egypt
```

CODE:

```
print("\n \"Hello \"'+name+"\', How Are You Doing \\ \n \"\"\" Your Age Is \"" +age+ "\"\" + \n And Your Country Is: "+live)
```

Example:
Remove the extra signs from the word and only the word remains. See the example to see the idea

OUTPUT:
name = "#@#@Jawabreh#@#@"

CODE:
```
name = "#@#@Jawabreh#@#@"
clean_name = name.strip("#@")
print(cleaned_name)
```

Example:
Create a variable that contains any number you want and its type is String, then put zeros before any number that is put as the value of the variable, provided that the width of the numbers does not exceed 4 numbers. For example, 20 is 0020, 199 is 0199, and 1200 is the same as 1200. See what is required in the following example.

OUTPUT:
0009
0015
0130
0950
1500

CODE:
```
num1 = "9"
num2 = "15"
num3 = "130"

print(num1.zfill(4))
print(num2.zfill(4))
print(num3.zfill(4))
```

Example:
Place @ signs before any String you are given, provided that the number of characters does not exceed 20. See the example to see the idea.

OUTPUT:
@@@@@@@@@@@@@@Ahmad
@@@@@@@@@@@@Jawabreh

CODE:
```
first_name = "Ahmad"
surname = "Jawabreh"
print(first_name.rjust(20,'@'))
print(surname.rjust(20,'@'))
```

Example:
Convert uppercase letters to lowercase letters and vice versa. See the example to see the idea
first_name = "AhMaD"
surname = "jAwAbreh"

OUTPUT:
aHmAd
JaWaBREH

CODE:
```
first_name = "AhMaD"
surname = "jAwAbreh"
print(first_name.swapcase())
print(surname.swapcase())
```

Example:
Count how many times the word Love is repeated in the String that will be given to you.
msg = "I Love Python And Although Love Elzero Web School"

CODE:
```
msg = "I Love Python And Although Love Elzero Web School Love"
print(msg.count("Love"))
```

Example:
Print the index for the letter w in the word Jawabreh

CODE:

```
msg = "Jawabreh"
print(msg.index('w'))
```

Example:
Replace the following word “<3” with the word Love only once in the sentence that will be given to you
msg = "I <3 Python And Although <3 Elzero Web School"

CODE:

```
msg = "I <3 Python And Although <3 Elzero Web School"
print(msg.replace("<3", "Love", 1))
```

Example:
Create three variables containing your name, age, and country. Make sure that the age is an Integer and not a String, and print the following message using the new Format method “f.” See the example.
My Name Is Ahmad, And My Age Is 22, and My Country is Palestine

CODE:

```
name = "Ahmad"
age = 21
country = "Palestine"
output = f"My Name Is {name}, And My Age Is {age}, and My Country is {country}"
print(output)
```

9.0 Numbers in Python:

Numbers in Python	
Integers	x = 5, y = -10, z = 0
Float	a = 3.14, b = -2.5, c = 1.0e5
Complex	comp1 = 2 + 3j, comp2 = -1.5 - 2j

figure 2: Numbers in Python

- Integers are whole numbers without a fractional component.
- Floats are numbers with a decimal point or in exponential form.
- Complex numbers have a real and an imaginary part, expressed as a + bj, where a and b are real numbers, and j is the imaginary unit.
-

Example:
Convert the number 10 to a Floating Point Number with ten digits after the decimal point

CODE:

```
num = 10

formatted_float = format(float(num), ".10f")
print(formatted_float)
print(type(float(num)))
```

Example:
Convert the number 159.650 to Integer, then print it in the first line, then in the second line you print its type and confirm that it is Integer.

CODE:

```
num = 159.650
int_num = int(num)
print(int_num)
print(type(int_num))
```

10.0 Arithmetic Operations:

Python's Arithmetic Operations	
Addition	print(10+5), print(-16.5+-4.2) print(31.2+-5)
Subtraction	print(10-5), print(-16.5--4.2) print(31.2--5)
Multiplication	print(10*5), print(-16.5*-4.2) print(31.2*-5)
Division	print(10/5), print(-16.5/-4.2) print(31.2/-5)
Modulus	print(10%5)
Exponent	print(5**2), print(-16.5**7) print(10**10)
Floor Division	print(100//20), print(119//20) print(120//20)

figure 3: Arithmetic Operations

11.0 String Indexing and Slicing:

In Python, strings are zero-indexed, meaning the first character is at index 0, the second at index 1, and so on. You can use square brackets to access individual characters in a string. On the other hand, slicing allows you to extract a portion of a string. The syntax is string[start:stop], where start is the index of the first character you want, and stop is the index of the first character you don't want to include.

Indexing:

```
my_string = "Hello, World!"
print(my_string[0])  # Output: H
print(my_string[7])  # Output: W
print(my_string[-1])  # Output: !
```

Slicing:

```
my_string = "Hello, World!"
print(my_string[0:5])  # Output: Hello
print(my_string[7:])   # Output: World!
print(my_string[:5])   # Output: Hello
print(my_string[:-1])  # Output: Hello, World
```

11.1 String Indexing and Slicing Examples:

Example:
Create a variable (name) with the value “Jawabreh” inside it, then using Indexing + Slicing, bring the second letter in the first line, the third letter in the second line, and the last letter in the third line. You must bring the letter in a dynamic way, as the word can change.

CODE:
```
name = "Jawabreh"
print(name[0])
print(name[1])
print(name[-1])
```

Example:
Create a variable (name) with the value “Elzero” inside it, then using Indexing + Slicing, create a code to show the below output

OUTPUT:
"lze"
"Ezr"
"rzE"

CODE:
```
name = "Elzero"
print(name[1:4] )
print(name[0::2])
print(name[-2::-2])
```

12.0 Lists:

- List items are enclosed in square brackets.
- Lists are ordered.
- Lists are mutable, you can add, delete, and edit the list easily
- List items are not unique, which means you can repeat the same item more than once.
- Lists allows you to use different data types for the items.
- The index of the first item is 0.
- You are allowed to use back indexing (negative indexing) ex: -1, -2 etc..
- You are allowed to use steps in the lists.

12.1 List Methods:

1-) append(): append (add) new item to the list

Example:
```
myList =[1,2,3,4]
myList.append(5)
print(myList)
```

OUTPUT:
```
[1, 2, 3, 4, 5]
```

Note: we can append a list to the main list, and the list that has been appended will be identified as one item.

Example:
```
myList =[1,2,3,4]
aList = [5,6,7]
myList.append(aList)
print(myList)
```

OUTPUT: `[1, 2, 3, 4, [5 , 6, 7]]`

How can we access an item in our list?
Let's assume that we want to print the item that has index 2
```
print(myList[2])
```

OUTPUT: `3`

2-) extend(): extend the list by appending new list in the main list, in the append() function it's appending the new list to the main one but the new list will be identified as one item, here we are extending the main list by appending the items of the new list in the main list.

Example:

```
myList =[1,2,3,4]
aList = [5,6,7]
myList.extend(aList)
print(myList)
```

OUTPUT:

[1, 2, 3, 4, 5 , 6, 7]

We can extend append more than one list items in the main list

Example:

```
myList =[1,2,3,4]
aList = [5,6,7]
bList = [8, 9, 10]
myList.extend(aList)
myList.extend(bList)
print(myList)
```

OUTPUT: [1, 2, 3, 4, 5 , 6, 7, 8, 9, 10]

3-) remove(): remove an item from the list

Example:

```
myList =[1,2,3,4]
myList.remove(2)
print(myList)
```

OUTPUT: [1, 3, 4]

Note: if the value that we want to delete is repeated in the list so the function will delete only the first occurrence of the value.

4-) sort(): order the list items

Example:
```
myList =[1, 5, 3, 3, 2]
myList.sort()
print(myList)
```

OUTPUT: [1, 2, 3, 4, 5]

We can also make the function ordering the items reversed

Example:
```
myList =[1, 5, 3, 4, 2]
myList.sort(reverse = True)
print(myList)
```

OUTPUT: [5, 4, 3, 2, 1]

5-) reverse(): reverse the list items upside down
Example:
```
myList =[1, 5, 3, 4, 2]
myList.sort(reverse = True)
print(myList)
```

OUTPUT: [2, 4, 3, 5, 1]

6-) clear(): delete all the items of the list
Example:
```
myList =[1, 5, 3, 4, 2]
myList.clear()
print(myList)
```

OUTPUT: []

7-) copy(): copy the list items
Example:
```
myList =[1, 5, 3, 4, 2]
aList = myList.copy()
print(aList)
```
OUTPUT: [1, 5, 3, 4, 2]

8-) count(): count how many times the item is repeated
Example:
myList =[1, 5, 1, 4, 1]
print(myList.count(1))

OUTPUT: 3

9-) index(): get the index of an item by providing it's value
Example:
myList =[1, 5, 1, 4, 1]
print(myList.index(1))

OUTPUT: 0

Note: if the value is repeated in the list, you will get the index of the first occurrence of the value.

10-) pop(): remove and return the last item from a list
Example:
myList =[1, 5, 1, 4, 1]
print(myList.pop(1))

OUTPUT: 5

11-) insert(): insert (add) new item to the list by specifying the index where you want to insert the item

Example:
myList =[1, 5, 1, 4, 1]
print(myList.insert(1, 9))

OUTPUT: [1, ,9, 5, 1, 4, 1]

Example:
myList =[1, 5, 1, 4, 1]
print(myList.insert(-1, 9))

OUTPUT: [1, 5, 1, 4, 1, 9]

But what is the difference between insert() and append() if both of them are adding new item/s to the list ?

append(): insert new value to the end of the list
insert(): insert new value to anywhere in the list

12.2 List Examples:

Example:
Make a list containing the names of your friends, with at least 5 names. In the first and second lines, you are required to print the name of the first friend in the list in two ways, then in the third and fourth lines, you print the name of the last friend in the list in two ways.

CODE:
```
# We Will use index and negative index to do it
friends = ["Ahmad", "Anas", "Mohammad", "Hussain", "Abd"]
print(friends[0])
print(friends[-len(friends)+1])
print(friends[-1])
print(friends[len(friends)-2])
```

OUTPUT:
```
Ahmad
Anas
Abd
Hussain
```

Example:
From the previous list, print the names that contain an odd index on the first line, and in the second line, print the names that contain an even index.

CODE:
```
friends = ["Osama", "Ahmed", "Sayed", "Ali", "Mahmoud"]
print(friends[1::2])
print(friends[::2])
```

Example:
Print the List of names Number 2, 3, and 4 on the first line, then the last name and the one before it on the second line, knowing that the code must work if we change the number of items in the list.

CODE:

```
friends = ["Osama", "Ahmed", "Sayed", "Ali", "Mahmoud"]
print(friends[1:4])
print(friends[-3:-1])
```

Example:
Update the last two names in the list to “Elzero”

CODE:

```
friends = ["Osama", "Ahmed", "Sayed", "Ali", "Mahmoud"]
friends[-2:] = ["Elzero", "Elzero"]
print(friends)
```

Example:
Add a name from your friends to the list at the beginning of the list first, then add another name at the end of the list

CODE:

```
friends = ["Osama", "Ahmed", "Sayed", "Ali", "Mahmoud"]

friends.append("Amer")
#OR
friends.insert(len(friends),"Amer")

friends.insert(0,"Musa")
print(friends)
```

Example:
Delete the first two names from the list, then end on another line and delete the last two names from the list

CODE:

```
friends = ["Nasser", "Osama", "Ahmed", "Sayed", "Salem"]

del friends[:2]
print(friends)

del friends[-1]
print(friends)
```

Example:
Create two more lists with more friends, then add them to the first list to come up with a final list with all your friends

CODE:

```
friends = ["Ahmed", "Sayed"]
employees = ["Samah", "Eman"]
school = ["Ramy", "Shady"]

friends.extend(employees)
friends.extend(school)
print(friends)
```

Example:
Arrange the names in the list on the first line from A to Z and on the second line from Z to A

CODE:

```
friends = ["Ahmed", "Sayed", "Samah", "Eman", "Ramy", "Shady"]

a_to_z = sorted(friends)
z_to_a = sorted(friends, reverse=True)

print(a_to_z)
print(z_to_a)
```

Example:
Count the number of friends in the list

CODE:

```
friends = ["Ahmed", "Sayed", "Samah", "Eman", "Ramy", "Shady"]

print(len(friends))
```

Example:
Make a list containing the famous programming languages and within it a submenu containing the names of famous frameworks. Then, in the first line, print the name of the first framework in the submenu, and in the second line, the name of the last framework in the submenu, taking into account that the submenu can increase, but it is always the last item. In the main menu

CODE:

```
technologies = ["Html", "CSS", "JS", "Python", ["Django", "Flask", "Web"]]

frameworks = technologies[-1]
print((frameworks[0]))
print((frameworks[-1]))
```

13.0 Tuples

- Tuple items are enclosed in parentheses
- You can remove the parentheses if you want
- Tuples are ordered, so you can use the indexing to access any item
- Tuples are immutable, you can't add or delete items
- Tuple items are not unique
- Tuples may contain different data types at the same time
- Operators that is used in String and Lists are available in the Tuples
- If you have only one item in you tuple, use comma after the item so the compiler can understand that this is tuple

13.1 Tuples Concatenation :

Example:
a = (1, 2, 3, 4, 5)
b = (6, 7, 8, 9, 10)
c = a + b
print(c)
OUTPUT: (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

Example:
a = (1, 2, 3, 4, 5)
b = (6, 7, 8, 9, 10)
c = a + ("Hello", "World") +b
print(c)
OUTPUT: (1, 2, 3, 4, 5, "Hello", "World", 6, 7, 8, 9, 10)

13.2 Repetition :

If you want to repeat same Tuple more than one time you can multiply it with the number of the repetitions you want

Example:
a = (1, 2)
print(a*3)\

OUTPUT: (1, 2, 1, 2, 1, 2)

Note: same idea works with Tuples, Lists and Strings.

13.3 Tuples Methods：

1-) count(): counts the repetition of a specific item (element)

Example:
a = (1, 2, 2, 2, 5)
print(a.count(3))

OUTPUT: 3
2-) index(): get the index of a specific element by providing it's value

Example:
a = (1, 2, 2, 2, 5)
print(a.count(1))

OUTPUT: 0

3-) index(): get the index of a specific element by providing it's value

Example:
a = (1, 2, 2, 2, 5)
print(a.count(1))

OUTPUT: 0

13.3 Tuples Examples：

Example:
Create a tuple consisting of one element, and let it be your name without using brackets (), then print the only element in the tuple on the first line, and on the second line, print the type to make sure that the type is tuple.

CODE:
```
myName = "Ahmad",
print(myName)
print(type(myName))
```

Example:
Create a tuple containing the names of 3 of your friends and the first name is Osama. Use your experience and what you learned previously to change the first name from Osama to Elzero. Print the content of the tuple on the first line. Print the type to make sure it is a tuple and not another data type, on the line Third, print the number of elements within the tuple

CODE:
```
names = ("Osama", "Ahmad", "Jawabreh")

names = ("Elzero",) + names[1:]
print(names)
print(type(names))
print(len(names))
```

Example:
Create a tuple containing the numbers 1 to 3. Create a tuple containing the letters A to C. Concatenate them into a new tuple and print its content in the first line. In the second line, count the number of elements contained within the new tuple.

CODE:
```
nums = (1, 2, 3)
ltrs = ('A', 'B', 'C')
my_tuple = nums+ltrs
print(my_tuple)
print(len(my_tuple))
```

Example:
Create a tuple containing 4 elements of any data type you want. Destruct the tuple and make an Assign for the first value of the variable a, the second value of the variable b, and the fourth value of the variable c. Print the variables a, b, c each one on a different line. Make sure that you have Destruct with just one line

CODE:

```
nums = (1, 2, 3, 4)
# tuple destruction
a, b, _, c = nums
# print the values
print(a)
print(b)
print(c)
```

14.0 Sets:

- Set items are enclosed with curly braces
- Set items are not ordered and not indexed
- Set indexing and slicing are not allowed in the sets
- Set has only immutable data types (string, numbers, tuples) dict and list are not allowed
- Set items are unique, so you are not allowed to repeat the same value more than once

14.1 Set Methods:

1-) clear(): delete all the items of the set (make it empty)
Example:
```
a = {1,2,3,4,5}
print(a.clear())
```

OUTPUT: set()
2-) union(): combine elements from two or more sets and create a new set that contains all unique elements from the input sets.

Example:
```
a = {1,2,3,4,5}
b = {6,7,8}
c = {9,10}
print(a.union(b, c))
```

OUTPUT: {1,2,3,4,5,6,7,8,9,10}

3-) add(): add an element to a set
Example:
```
a = {1,2,3,4,5}
print(a.add(6))
```

OUTPUT: {1,2,3,4,5,6}

4-) copy(): create a shallow copy of a set.
Example:
```
a = {1,2,3,4,5}
b = a.copy()
print(a)
print(b)
```

OUTPUT: {1,2,3,4,5} {1,2,3,4,5}

5-) remove(): remove specific element from a set
Example:
a = {1,2,3,4,5}
print(a.remove(1))

OUTPUT: {2,3,4,5}
Note: if the element is not available an error will be printed

6-) discard(): remove specific element from a set
Example:
a = {1,2,3,4,5}
print(a.remove(1))

OUTPUT: {2,3,4,5}
Note: if the element is not available there will be no errors.

7-) pop(): remove random element from a set
Example:
a = {1,2,3,4,5}
print(a.pop())

OUTPUT: {2,3,4,5}

8-) update(): update a set with the elements of another iterable or with multiple elements
Example:
a = {1,2,3,4,5}
b = {1,6,7}
print(a.update(b))

OUTPUT: {1,2,3,4,5,6,7}

9-) difference(): get the difference between two set
Example:
a = {1,2,3,4,5}
b = {1,2,3}
print(a.difference(b))

OUTPUT: {4,5}

10-) difference_update(): remove the common elements between two sets.

Example:

```
a = {1,2,3,4,5}
b = {1,2,3}
a.difference_update(b)
print(a)
```

OUTPUT: {4,5}

11-) intersection(): get the common elements between two or more sets.

Example:

```
a = {1,2,3,4,5}
b = {1,2,3}
print(a.intersection(b))
```

OUTPUT: {1,2,3}

12-) intersection_update(): update a set with the intersection of itself and another iterable.

Example:

```
a = {1,2,3,4,5}
b = {1,2,3}
a.intersection_update(b)
print(a)
```

OUTPUT: {4,5}

13-) symmetric_difference(): the symmetric difference between two sets is the set of elements that are in either of the sets, but not in both.

Example:

```
a = {1, 2, 3, 4, 5}
b = {3, 4, 5, 6, 7}
print(a.symmetric_difference(b))
```

OUTPUT: {1,2,6,7}

14-) symmetric_difference_update(): to update a set with the symmetric difference of itself and another iterable.

Example:
```
a = {1, 2, 3, 4, 5}
b = {3, 4, 5, 6, 7}
a.symmetric_difference(b)
print(a)
```

OUTPUT: {1,2,6,7}

15-)issuperset(): used to check if a set is a superset of another set or any iterable. A superset is a set that contains all the elements of another set, and possibly more

Example:
```
a = {1, 2, 3, 4, 5}
b = {1,2,3}
print(a.issuperset(b))
```

OUTPUT: True

16-)issubset(): method is used to check if a set is a subset of another set or any iterable. A subset is a set that contains all the elements of another set

Example:
```
a = {1, 2, 3, 4, 5}
b = {1,2,3}
print(a.issubset(b))
```

OUTPUT: False

17-) isdisjoint(): method is used to check whether two sets are disjoint, meaning that they have no elements in common

Example:
```
a = {1, 2, 3, 4, 5}
b = {1,2,3}
print(a.isdisjoint(b))
```

OUTPUT: Fals

14.2 Set Examples:

Example:
Create a List containing the following numbers 1, 2, 3, 3, 4, 5, 1. Create a variable named unique_list and then store only the unique values from the previous List. In the first line, print the content of unique_list and make sure it contains the unique numbers. Only in the second line, print the unique_list type and make sure that the data type is a list. In the third line, print the unique_list without the last element.

CODE:

```
my_list = [1, 2, 3, 3, 4, 5, 1]

# the set items are unique but the list items are not
# so we can convert the list to set then to list to get to unique items

unique_set = set(my_list)
unique_list = list(unique_set)

print(unique_list)
print(type(unique_list))
print(unique_list[0:-1])
```

Example:
Create a new Set containing the numbers 1, 2, 3, then create a new Set containing the letters A, B, C, combine the first and second Set in three different ways and print each method on a line

CODE:

```
nums = {1, 2, 3}
ltrs = {'A', 'B', 'C'}

union = nums.union(ltrs)
nums.update(ltrs)
combined = nums | ltrs

print(union)
print(nums)
print(combined)
```

Example:
Create a Set containing elements 1, 2, 3 , create a second Set containing 1, 2, 3, 4, 5, 6 ,check whether all the contents of the first Set are present in the second or not

CODE:

```
set_one = {1, 2, 3}
set_two = {1, 2, 3, 4, 5, 6}

result = set_one.issubset(set_two)
print(result)
```

15.0 Dictionaries:

- Dictionary values are enclosed with curly braces
- Dictionary contains KEY : VALUE
- Dictionary key must be immutable (number, string, tuple), lists are not allowed
- Dictionary value can be any data type
- Dictionary key must be unique
- Dictionary is not ordered, you can access the elements by it’s key
- To print the dictionary (keys and values) we can use, print(dictname)
- To print all of the dictionary keys, print(dictionary.keys())
- To print all of the dictionary values, print(dictionary.values())
- To access element by it’s key, (dictname[key])

15.1 Two Dimensional Dictionaries:

a two-dimensional dictionary is essentially a dictionary of dictionaries. You can create it using a nested dictionary.

Example:

```
two_dimensional_dict = {
    'row1': {'column1': 1, 'column2': 2, 'column3': 3},
    'row2': {'column1': 4, 'column2': 5, 'column3': 6},
    'row3': {'column1': 7, 'column2': 8, 'column3': 9}
}

# Accessing elements
print(two_dimensional_dict['row1']['column2'])
```

15.2 Dictionary Methods:

1-)clear(): clear or remove all of the items

Example:

```
users = {“Name” : “Ahmad”}
print(users.clear())
```

OUTPUT: {}

2-)update(): add new item to the dictionary

Example:

```
users = {“Name” : “Ahmad”}
users.update({“Last Name” : “Jawabreh”})
print(users)
```

OUTPUT: {“Name” : “Ahmad”, “Last Name” : “Jawabreh”}

OR

```
users = {“Name” : “Ahmad”}
users[“Last Name”] = “Jawabreh”
```

3-) copy(): copy the items into new dictionary

Example:
users = {“Name” : “Ahmad”}
members = users.copy()
print(members)

OUTPUT: {“Name” : “Ahmad”}

4-) setdefault(): allows you to set the default value for a key in a dictionary. If the key is present in the dictionary, it returns the corresponding value. If the key is not present, it inserts the key with the specified default value.

Example:
```
my_dict = {'a': 1, 'b': 2, 'c': 3}
value = my_dict.setdefault('d', 0)
print(my_dict)
print(value)
```

OUTPUT:
```
{'a': 1, 'b': 2, 'c': 3, 'd': 0}
0
```

5-) popitem(): removes and returns the last key-value pair from the dictionary as a tuple

Example:
```
my_dict = {'a': 1, 'b': 2, 'c': 3}
key, value = my_dict.popitem()
print(f"Popped item: {key}: {value}")
print(f"Updated dictionary: {my_dict}")
```

OUTPUT:
```
Popped item: c: 3
Updated dictionary: {'a': 1, 'b': 2}
```

6-) items(): returns a view object that displays a list of dictionary's key-value tuple pairs

Example:
users = {“Name” : “Ahmad”}
print(users.items())
OUTPUT:
dict_items([(‘name’, ‘Ahmad’])

7-) fromkeys(): creates a new dictionary with keys from a given iterable (such as a list or tuple) and assigns a specified value to all keys.

Example:
keys = ['a', 'b', 'c']
default_value = 0
new_dict = dict.fromkeys(keys, default_value)
print(new_dict)
OUTPUT:
{'a': 0, 'b': 0, 'c': 0}

14.3 Dictionar Examples:

Example:
Create a Dictionary containing three programming skills with the percentage of your level in them Without using the loop, print each skill on a line with the level written in percentage next to it Add a new skill to the Dictionary with its percentage and print it on the fifth line

CODE:

```
programming_skills = {
    "HTML": 90,
    "CSS": 80,
    "Python": 30
}

print(f"HTML Progress Is {programming_skills['HTML']}%")
print(f"CSS Progress Is {programming_skills['CSS']}%")
print(f"Python Progress Is {programming_skills['Python']}%")

programming_skills["AI"] = 20
print(f"AI Progress Is {programming_skills['AI']}%")
```