LISTS II

The *for* Loop, Lists, and Tuples

The *for* Loops (1 of 2)

- A *for* loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).
- A loop, one of the important structure in programming, is a part of program that can execute a block of code repeatedly.
- When designing programs you think ahead of times how many times the loop needs to be preformed.

The *for* Loops (2 of 2)

- Syntax:

```
for var in sequence:
    indented block of statements
```

- Sequence can be
  - Arithmetic progression of numbers
  - String
  - List
  - File object

Looping Through a String

Ex1_for_string_list.py

- Even strings are iterable objects, they contain a sequence of characters:

```
>>> for x in “py”:
        print(x)

p
y
```

```
>>> fruit = “apple”
>>> for x in fruit:
>>>     print(x)
a
p
p
l
e
```

Looping Through a List

- You can loop through the list items by using a *for* loop.
- Print all items in the list, one by one:

```
>>> fruits = ["apple", "banana", 'python programming',]
>>> for item in fruits:
        print(item)
apple
banana
python Programming
```

Indentation

- Where in other programming languages the indentation in code is for readability only, the **indentation** in Python is **very important**. Python uses indentation to indicate a block of code.
- Python relies on indentation (whitespace at the beginning of a line) to define **scope** in the code. Other programming languages often use curly-brackets for this purpose.
- Sometimes multiple instructions needs to be executed if a condition was true, the indentation level determine where the if ends.

The *range()* Function (1 of 3)

Ex2_list_range.py

- To loop through a set of code a specified number of times, we can use the *range*() function.
- The *range*(n) function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and counting from 0 to n-1.

```
>>> for x in range(3):
        print(x)

0
1
2
```

The *range()* Function (2 of 3)

- The *range*() function defaults to 0 as a starting value, however it is possible to specify the **starting** value by adding a parameter.

```
>>> for x in range(3, 10):
        print(x)

3
4
5
6
7
8
9
```

`range(3, 10)` generates the sequence $3, 4, 5, 6, 7, 8, 9$.

`range(0, 4)` generates the sequence $0, 1, 2, 3$.

`range(-4, 2)` generates the sequence $-4, -3, -2, -1, 0, 1$.

The *range()* Function (3 of 3)

- The *range*() function defaults to increment the sequence by 1, however it is possible to specify the step value by adding a **third** parameter.

```
>>> for x in range(3, 10, 2):
        print(x)

3
5
7
9
```

`range(3, 10, 2)` generates the sequence $3, 5, 7, 9$.
`range(0, 24, 5)` generates the sequence $0, 5, 10, 15, 20$.
`range(-10, 10, 4)` generates the sequence $-10, -6, -2, 2, 6$.

else in the *for* Loop

- The *else* keyword in a *for* loop specifies a block of code to be executed when the loop is finished.

```
>>> for x in range(5):
        print(x, end = ' ')
    else:
        print("Finally finished!")

0 1 2 3 4
Finally finished!
```

List Comprehensions

Ex3_List_Comprehension_Slice.py

- Combines the ***for*** loop and the creation of new elements of the **list** into **one** line

```
>>> squares = [value**2 for value in range(1, 11)]

[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

List Slicing

- Use slicing to loop through a subset of the elements in a list

```
>>> for i in squares[:5]:
        print(f"Prefix-{i}", end = " | ")
```

Prefix-1 | Prefix-4 | Prefix-9 | Prefix-16 | Prefix-25 |

- Use slicing to copy a list

```
>>> x = squares[:]
```

PE5_1

1. A – H, determine the output displayed by the lines of code. I – K, write the code by using the output. Save your code as *PE5_1.py*.

A	`a = list(range(5))`
	`print(a)`
		B	`b = []`
			`for i in range (5):`
			`  b.append(i)`
			`print(b)`

Output		Output	

C	`x = list(range(-10, 10))`
	`print(x)`
	`print(min(x), max(x), sum(x))`		

Output			
D	`even_num = list(range(2, 11, 2))` `print(even_num[0], even_num[-1])`		
Output			

E	`#Print all the odd numbers from 1 to 9 inclusive in a list, odd_num.`
Output	[1, 3, 5, 7, 9]
F	`# Make a list of the first 10 cubes and use a for loop to print out the value of each cube in a new line (see output below).`
Output	1 8 27 64 125 216 343 512 729 1000
G	`#Use a list comprehension to generate a list of the first 10 cubes. Use a for loop to print out the value of each cube in a row separated by a '|' (see output below).`
Output	1|8|27|64|125|216|343|512|729|1000|

PE5_2

2. List slicing. Save it as *PE5_2.py*.
a) Use a **list comprehension** to generate a list of all even numbers from 0 to 100 **inclusive**.
b) Use slicing to print the first five even numbers in the list.
c) Use slicing to print the last five even numbers in the list.
d) Use slicing to print all list numbers between 20 and 30 inclusive.

Example Output

```
[0, 2, 4, 6, 8]
[92, 94, 96, 98, 100]
[20, 22, 24, 26, 28, 30]
```

The Tuple Object (1 of 3)

Ex4_tuples.py

- Tuples, like lists, are ordered sequences of items
- Difference - tuples cannot be modified in place
  - *Have no **append**, **extend**, or **insert** method*
- Items of tuple cannot be directly deleted, sorted, or altered

The Tuple Object (2 of 3)

- All other list functions and methods apply
  - *Items can be accessed by indices*
  - *Tuples can be sliced, concatenated, and repeated*
- Tuples written as comma-separated sequences enclosed in parentheses
  - *Can also be written without the parentheses.*

The Tuple Object (3 of 3)

- Tuples have several of same functions as lists

```
t = 5, 7, 6, 2
print(t)
print(len(t), max(t), min(t), sum(t))
print(t[0], t[-1], t[:2])
```

[Run]

```
(5, 7, 6, 2)
4 7 2 20
5 2 (5, 7)
```

Indexing, Deleting, Slicing and Out of Bounds (1 of 2)

- Python does not allow out of bounds indexing for individual items in lists and tuples
  - *But does allow it for slices*

- Given

```
t = 5, 7, 6, 2
```

✘ `print(t[7])`
✘ `print(t[-7])`
✘ `del t[7]`

✓ `print(t[-7:7])`
✓ `print(t[-7:2])`
✓ `del t`

Indexing, Deleting, and Slicing Out of Bounds (2 of 2)

- If left index in slice too far negative
  - *Slice will start at the beginning of the list*
- If right index is too large,
  - *Slice will go to the end of the list.*

- `t = 5, 7, 6, 2`
- `t[-7:7] is (5, 7, 6, 2)`
- `t[-7:2] is (5, 7)`

PE5_3 & PE5_4 & PE5_5

- Write your codes and run

Lists & Tuples - Terminologies

- 1 append
- 2 clear
- 3 copy
- 4 extend
- 5 find
- 6 join
- 7 insert
- 8 pop
- 9 remove
- 10 sort
- 11 split
- 12 del
- 13 for
- 14 len
- 15 List
- 16 max
- 17 min
- 18 range
- 19 reverse
- 20 sorted
- 21 tuple
- 22 Loop
- 23 Iteration
- 24 Indentation
- 25 Immutable
- 26 List Comprehension
- 27 Mutable
- 28 Objects
- 29 Slicing
- 30 Out of Bounds
- 31 Zero Indexing

Quiz 5

- Quiz 5A has 10 questions in 15 minutes, 10 pts
  - *10 multiple choice/true or false questions, 1 pt. for each question*
  - *Quiz 5A has two attempt, the higher grade will be selected*
  - *Submit Quiz 5A (at least 1-minute) **before** the due time to Blackboard*

- Quiz 5B has 2 code questions, 15 pts
  - *Write the Python code based on the given question*
  - *Each question will be given during the last 10-minute of each session of week 5*
  - *Quiz 5B-1 on session A, and Quiz 5B-2 on session B*
  - *Quiz 5B has one attempt*

DB 5

- Instruction:
  1) Choose any **one** of the questions from **PE5_1**, any **one** of the questions from **PE5_6**, **and** any **one** of the questions from **PE5_7**. Please **avoid** selecting the exact same questions. Make sure to indicate the **question #** you're working on in the thread title as soon as you open your thread. Then you can **explain and edit your questions** (1.2 pt).
  2) Explain the following (0.3 pt).
  Why can't you use the *sort* method in an assignment statement?
  Ex., inventory = items.sort()
  3) Submit your posts before the due date. Let's learn from each other.