Experiment 2: Explore functions, list, tuples & ranges in Python.

Requirement: Laptop or Desktop with Python installed

Theory

Creating Tuples

Tuples are a type of sequence, like strings. But unlike strings, which can only contain characters, tuples can contain elements of any type. That means you can have a tuple that stores a bunch of high scores for a game, or one that stores a group of employee names. But tuple elements don't have to all be of the same type. You could create a tuple with both strings and numbers, if you wanted. And you don't have to stop at strings and numbers. You can create a tuple that contains a sequence of graphic images, sound files, or even a group of aliens (once you learn how to create these things, which you will in later chapters). Whatever you can assign to a variable, you can group together and store as a sequence in a tuple.

CREATING FUNCTIONS

You've already seen several built-in functions in action, including len() and range(). Well, if these aren't enough for you, Python lets you create functions of your very own. Your functions work just like the ones that come standard with the language. They go off and perform a task and then return control to your program. Creating your own functions offers you many advantages. One of the biggest is that it allows you to break up your code into manageable, bite-sized chunks. Programs that are one, long series of instructions with no logical breaks are hard to write, understand, and maintain. Programs that are made up of functions can be easier to create and work with. Just like the functions you've already met, your new functions should do one job well.

Defining a Function

I began the definition of my new function with a single line:

def instructions():

This line tells the computer that the block of code that follows is to be used together as the function instructions(). I'm basically naming this block of statements. This means that whenever I call the function instructions() in this program, the block of code runs. This line and its block are a function definition. They define what the function does, but don't run the function. When the computer sees the function definition, it makes a note that this function exists so it can use it later. It won't actually run the function until it sees a function call for it, later in the program. To define a function of your own, follow my example. Start with def, followed by your function name, followed by a pair of parentheses, followed by a colon, and then your indented block of statements. To name a function, follow the basic rules for naming variables. Also, try to use a name that conveys what the function produces or does.

Documenting a Function

Functions have a special mechanism that allows you to document them with what's called a docstring (or documentation string). I created the following docstring for instructions():

```
"""Display game instructions."""
```

A docstring is typically a triple-quoted string and, if you use one, must be the first line in your function. For simple functions, you can do what I did here: write a single sentence that describes what the function does. Functions work just fine without docstrings, but using them is a good idea. It gets you in the habit of commenting your code and makes you describe the function's one, well-defined job. Also, a function's docstring can pop up as interactive documentation while you type your call to it in IDLE.

Understanding Abstraction

By writing and calling functions, you practice what's known as abstraction. Abstraction lets you think about the big picture without worrying about the details. So, in this program, I can just use the function instructions() without worrying about the details of displaying the text. All I have to do is call the function with one line of code, and it gets the job done. You might be surprised where you find abstraction, but people use it all the time. For example, consider two employees at a fast food place. If one tells the other that he just filled a #3, and "sized it," the other employee knows that the first employee took a customer's order, went to the heat lamps, grabbed a burger, went over to the deep fryer, filled their biggest cardboard container with French fries, went to the soda fountain, grabbed their biggest cup, filled it with soda, gave it all to the customer, took the customer's money, and gave the customer change. Not only would this version be a boring conversation, but it's unnecessary. Both employees understand what it means to fill a #3 and "size it." They don't have to concern themselves with all the details because they're using abstraction.

Programs

Class Task-1

Craps Roller Program

```
# Craps Roller
# Demonstrates random number generation

import random

# generate random numbers 1 - 6
die1 = random.randint(1, 6)
die2 = random.randrange(6) + 1

total = die1 + die2

print("You rolled a", die1, "and a", die2, "for a total of", total)

input("\n\nPress the enter key to exit.")
```

Class Task-2

Password Program

```
# Password
# Demonstrates the if statement

print("Welcome to System Security Inc.")
print("-- where security is our middle name\n")

password = input("Enter your password: ")

if password == "secret":
    print("Access Granted")

input("\n\nPress the enter key to exit.")
```

Class Task-3

Granted or Denied Program

```
# Granted or Denied
# Demonstrates an else clause

print("Welcome to System Security Inc.")
print("-- where security is our middle name\n")

password = input("Enter your password: ")

if password == "secret":
    print("Access Granted")
else:
    print("Access Denied")

input("\n\nPress the enter key to exit.")
```

Class Task-4

Computer Mood Program

```
# Mood Computer
# Demonstrates the elif clause

import random

print("I sense your energy.  Your true emotions are coming across my
screen.")
print("You are...")

mood = random.randint(1, 3)

if mood == 1:
    # happy
    print( \
    """
       -----------
       |         |
       | O    O  |
       |   <     |
       |         |
       | .     . |
       |  `...`  |
       -----------
                   """)
elif mood == 2:
    # neutral
    print( \
    """
       -----------
       |         |
       | O    O  |
       |   <     |
       |         |
       | ------  |
       |         |
       -----------
                   """)
elif mood == 3:
    # sad
    print( \
    """
       -----------
       |         |
       | O    O  |
       |   <     |
       |         |
       |  .'.    |
       | '   '   |
       -----------
                   """)
else:
    print("Illegal mood value!  (You must be in a really bad mood).")

print("...today.")

input("\n\nPress the enter key to exit.")
```

Class Task-5

Three Year Old

```
# Three Year-Old Simulator
# Demonstrates the while loop

print("\tWelcome to the 'Three-Year-Old Simulator'\n")
print("This program simulates a conversation with a three-year-old
child.")
print("Try to stop the madness.\n")

response = ""
while response != "Because.":
    response = input("Why?\n")

print("Oh.  Okay.")

input("\n\nPress the enter key to exit.")
```

Class Task-6

Loopy String

```
# Loopy String
# Demonstrates the for loop with a string

word = input("Enter a word: ")

print("\nHere's each letter in your word:")
for let in word:
    print(let)

input("\n\nPress the enter key to exit.")
```

Class Task-7

Message Analyzer

```
# Message Analyzer
# Demonstrates the len() function and the in operator

message = input("Enter a message: ")

print("\nThe length of your message is:", len(message))

print("\nThe most common letter in the English language, 'e',")
if "efg" in message:
    print("is in your message.")
else:
    print("is not in your message.")

input("\n\nPress the enter key to exit.")
```

Class Task-8

Random Access

```
# Random Access
# Demonstrates string indexing

import random

word = "index hello"
print("The word is: ", word, "\n")

high = len(word)
low = -len(word)

for i in range(10):
    position = random.randrange(low, high)
    print("word[", position, "]\t", word[position])

input("\n\nPress the enter key to exit.")
```

Class Task-9

No Vowels

```
# No Vowels
# Demonstrates creating new strings with a for loop

message = input("Enter a message: ")
new_message = ""
VOWELS = "aeiou"

print()
for letter in message:
    if letter.lower() not in VOWELS:
        new_message += letter
        print("A new string has been created:", new_message)

print("\nYour message without vowels is:", new_message)

input("\n\nPress the enter key to exit.")
```

Class Task-10

Hero Inventory

```
# Hero's Inventory
# Demonstrates tuple creation

# create an empty tuple
inventory = ()

# treat the tuple as a condition
if not inventory:
    print("You are empty-handed.")

input("\nPress the enter key to continue.")


# create a tuple with some items
inventory = ("sword",
             "armor",
             "shield",
             "healing potion",
          5)

# print the tuple
print("\nThe tuple inventory is:")
print(inventory)

# print each element in the tuple
print("\nYour items:")
for item in inventory:
    print(item)

input("\n\nPress the enter key to exit.")
```

Class Task-11

Instruction Function

```
# Instructions
# Demonstrates programmer-created functions
def instructions():

     print(
     """
     Welcome to the greatest intellectual challenge of all time: Tic-
Tac-Toe.
     This will be a showdown between your human brain and my silicon
processor.
     You will make your move known by entering a number, 0 - 8. The
number
     will correspond to the board position as illustrated:
     0 | 1 | 2
     ---------
     3 | 4 | 5
     ---------
     6 | 7 | 8
     Prepare yourself, human. The ultimate battle is about to begin.
\n
     """
     )
# main
print("Here are the instructions to the Tic-Tac-Toe game:")
instructions()
print("Here they are again:")
instructions()
print("You probably understand the game by now.")
input("\n\nPress the enter key to exit.")
```

Class Task-12

Receive and Return

```
# Receive and Return
# Demonstrates parameters and return values
def display(message):
     print(message)
def give_me_five():
     five = 5
     return five
def ask_yes_no(question):
     """Ask a yes or no question."""
     response = None
     while response not in ("y", "n"):
          response = input(question).lower()
     return response
# main
display("Here's a message for you.\n")
number = give_me_five()
print("Here's what I got from give_me_five():", number)
answer = ask_yes_no("\nPlease enter 'y' or 'n': ")
print("Thanks for entering:", answer)
input("\n\nPress the enter key to exit.")
```

Class Task-13

Birthday Wishes

```
# Birthday Wishes
# Demonstrates keyword arguments and default parameter values

# positional parameters
def birthday1(name, age):
     print("Happy birthday,", name, "!", " I hear you're", age,
"today.\n")
# parameters with default values
def birthday2(name = "Jackson", age = 1):
     print("Happy birthday,", name, "!", " I hear you're", age,
"today.\n")
birthday1("Jackson", 1)
birthday1(1, "Jackson")
birthday1(name = "Jackson", age = 1)
birthday1(age = 1, name = "Jackson")

birthday2()
birthday2(name = "Katherine")
birthday2(age = 12)
birthday2(name = "Katherine", age = 12)
birthday2("Katherine", 12)
input("\n\nPress the enter key to exit.")
```