MORE FUNCTIONS

Structured Programming

Creating a Function

Ex1_Functions123.py

- Syntax:
```
def name_function(parameters):
    statements
    ............
```
- Function definitions cannot be empty, but if you for some reason have a function definition with no content, put in the **`pass`** statement to avoid getting an error.
- Example:

```
>>>def myfunction():
       pass
```

Scope of Variables (1 of 5)

- Variable created inside a function can only be accessed by statements **inside** that function
  - *Ceases to exist when the function is exited*
- Variable is said to be **local** to function or to have **local scope**
- If variables created in two different functions have the same name
  - *They have no relationship to each other*

Scope of Variables (2 of 5)

Ex2_scopeVariables.py

- Example:
  Variable x in the function main(), and variable x in the function trivial() are **different** variables

```
def main():
    ## Demonstrate the scope of variables.
    x = 2
    print(str(x) + ": function main")
    trivial()
    print(str(x) + ": function main")

def trivial():
    x = 3
    print(str(x) + ": function trivial")

main()
```

[Run]

```
2: function main
3: function trivial
2: function main
```

Scope of Variables (3 of 5)

Ex3_scopeVariables_error.py

- Example:
  - Variable x created in function main()
  - Not recognized by function trivial
- How to debug?

```
def main():
    ## Demonstrate the scope of local variables.
    x = 5
    trivial()

def trivial():
    print(x)

main()
```

Scope of Variables (4 of 5)

- Scope of a variable is the portion of the program that can refer to it
- To make a variable **global**, place assignment statement that creates it at **top** of program.
  - *Any function can read the value of a global variable*
  - *Value cannot be altered inside a function unless*

```
global globalVariableName
```

Scope of Variables (5 of 5)

Ex4_global_constant.py

- Example: Program contains a global variable

```
x = 0   # Declare a global variable.

def main():
    ## Demonstrate the scope of a global variable.
    print(str(x) + ": function main")
    trivial()
    print(str(x) + ": function main")

def trivial():
    global x
    x += 7
    print(str(x) + ": function trivial")

main()
```

[Run]

```
0: function main
7: function trivial
7: function main
```

Named Constants

- Employs a special constant used several times in program
- Convention programmers use
  - *Create a global variable*
  - *Name written in uppercase letters with words separated by underscore*
- In Python, programmer is responsible for not changing value of the variable

Functions Calling Other Functions (1 of 2)

Ex5_functionCalling.py

- Function can call another function
- When the called function terminates
  - *Control returns to the place in calling function just after where function call occurred.*

Functions Calling Other Functions (2 of 2)

- Example : Function firstPart calls the function secondPart.

```
def main():
    ## Demonstrate functions calling other functions.
    firstPart()
    print(str(4) + ": from function main")

def firstPart():
    print(str(1) + ": from function firstPart")
    secondPart()
    print(str(3) + ": from function firstPart")

def secondPart():
    print(str(2) + ": from function secondPart")

main()
```

[Run]

```
1: from function firstPart
2: from function secondPart
3: from function firstPart
4: from function main
```

PE10_1, PE10_2, and PE10_3

- Write your codes and run

Functions with Returning Values (1 of 2)

- Functions can return any type of objects

person.py

```python
def build_person(first_name, last_name, age=None):
    """Return a dictionary of information about a person."""
    person = {'first': first_name, 'last': last_name}
    if age:
        person['age'] = age
    return person

musician = build_person('jimi', 'hendrix', age=27)
print(musician)
musician = build_person('Adele', 'Adkins')
print(musician)
```

Python - person.py:12 ✔

```
{'first': 'jimi', 'last': 'hendrix', 'age': 27}
{'first': 'Adele', 'last': 'Adkins'}
```

greeter.py

```python
def get_formatted_name(first_name, last_name):
    """Return a full name, neatly formatted."""
    full_name = f"{first_name} {last_name}"
    return full_name.title()

# This is an infinite loop!
while True:
    print("\nPlease tell me your name:")
    print("(enter 'q' at any time to quit)")

    f_name = input("First name: ")
    if f_name == 'q':
        break

    l_name = input("Last name: ")
    if l_name == 'q':
        break

    formatted_name = get_formatted_name(f_name, l_name)
    print(f"\nHello, {formatted_name}!")
```

Functions with Returning Values (2 of 2)

Ex6_multiReturnValues.py

- Tuples as returned values

```
def total(w, x, y=10, z=20):
    return (w ** x) + y + z
```

```
# define a function with multiple return values in a tuple
def operations(x=1, y=1):
    return (x+y, x-y, x*y, x/y)

print(operations())
print(operations(5, 4))
```

Arbitrary Arguments

Ex7_pizza.py

- Example:
  If you do not know how many arguments that will be passed into your function, add a * before the parameter name in the function definition.

pizza.py

```python
def make_pizza(size, *toppings):
    """Summarize the pizza we are about to make."""
    print(f"\nMaking a {size}-inch pizza with the following toppings:")
    for topping in toppings:
        print(f"- {topping}")

make_pizza(12, "pepperoni")
make_pizza(16, "mushrooms", 'green peppers', 'extra cheese')
```

Python - pizza.py:6 ✔

```
Making a 12-inch pizza with the following toppings:
- pepperoni

Making a 16-inch pizza with the following toppings:
- mushrooms
- green peppers
- extra cheese
```

PE10_4, PE10_5, and PE10_6

- Write your codes and run

Arbitrary Keyword Arguments

Ex8_user_profile.py

- Using arbitrary keyword arguments to make a dictionary

user_profile.py

```python
def build_profile(first, last, **user_info):
    """Build a dictionary containing everything we know about a user."""
    user_info['first_name'] = first
    user_info['last_name'] = last
    return user_info

user_profile = build_profile('albert', 'einstein',
                             location='princeton',
                             field='physics')

print(user_profile)
```

Python - user_profile.py:11 ✔

```
{'location': 'princeton', 'field': 'physics', 'first_name': 'albert', 'last_name': 'einstein'}
```

Creating a Module (1 of 4)

- A module is a file with extension .py
  - *Contains functions and variables*
  - *Can be used (imported) by any program*
  - *can be created in I D E or any text editor*
  - *Looks like an ordinary Python program*
- To gain access to the functions and variables
  - *place a statement, import moduleName at the beginning of the program*

Creating a Module (2 of 4)

Ex9a_making_pizzas.py

- Save **pizza.py** in same folder as making_pizzas.py
- Import an entire module:

pizza.py | *making_pizzas.py*

```python
import pizza

pizza.make_pizza(16, 'pepperoni')
pizza.make_pizza(12, 'mushrooms', 'green peppers', 'extra cheese')
```

Python - making_pizzas.py:5 ✔

```
Making a 16-inch pizza with the following toppings:
- pepperoni

Making a 12-inch pizza with the following toppings:
- mushrooms
- green peppers
- extra cheese
```

Creating a Module (3 of 4)

Ex9b_making_pizzas.py

- Import a specific function ***from*** a module

pizza.py | making_pizzas.py

```python
from pizza import make_pizza

make_pizza(16, 'pepperoni')
make_pizza(12, 'mushrooms', 'green peppers')
```

Python - making_pizzas.py:5 ✔

```
Making a 16-inch pizza with the following toppings:
- pepperoni

Making a 12-inch pizza with the following toppings:
- mushrooms
- green peppers
```

Creating a Module (4 of 4)

Ex9c_making_pizzas.py

- Using **as** to give a function an **alias**

pizza.py | making_pizzas.py

```python
from pizza import make_pizza as mp

mp(16, 'pepperoni')
mp(12, 'mushrooms')
```

Python - making_pizzas.py:4 ✓

```
Making a 16-inch pizza with the following toppings:
- pepperoni

Making a 12-inch pizza with the following toppings:
- mushrooms
```

Top-Down Design (1 of 2)

- To make a complicated problem more understandable
  - *Divide it into smaller, less complex subproblems.*
  - *Called stepwise refinement*
- Top-down design and structured programming
  - *Techniques to enhance programming productivity*

Top-Down Design (2 of 2)

- Design should be easily readable and emphasize small module size.
- Tasks proceed from general to specific as you read down the chart.
- Subtasks should be single-minded.
- Subtasks should be independent of each other.

Functions Terminologies

- ❑ 1 Actual Parameters
- ❑ 2 Arbitrary Parameters
- ❑ 3 Arbitrary Keyword Arguments
- ❑ 4 Arguments
- ❑ 5 As
- ❑ 6 Built-in Functions
- ❑ 7 Call a Function
- ❑ 8 Define a Function
- ❑ 9 Default Parameter Value
- ❑ 10 Formal Parameters
- ❑ 11 From
- ❑ 11 Global/Local Variables
- ❑ 12 Keyword Arguments
- ❑ 13 Import
- ❑ 14 Modules
- ❑ 15 None
- ❑ 16 Optional Arguments
- ❑ 17 Parameters
- ❑ 18 Positional Arguments
- ❑ 19 Return Statement
- ❑ 20 Scope
- ❑ 21 Top-Down Design
- ❑ 22 Organization
- ❑ 23 Autonomy
- ❑ 24 Encapsulation
- ❑ 25 Reusability

Quiz 10

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

- Quiz 10B 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 10*
  - *Quiz 10B-1 on session A, and Quiz 10B-2 on session B*
  - *Quiz 10B has one attempt*

DB 10

- Instruction:
  1) Choose any **three terminologies** from the PowerPoint this week or last week. Please **avoid** selecting the exact same terms. Make sure to indicate the **terms** you're working on in the thread title as soon as you open your thread. Then you can **explain and edit your terms** (1.2 pt).
  2) There are **three** ways to **pass arguments** to parameters in a function. List them and give an example of each type (0.3 pt).
  3) Submit your posts before the due date. Let's learn from each other.