CS PROFESSIONAL ELECTIVE | FINAL ASSIGNMENT #1

MATILLA, CARL ANDRIE D. | BSCS 3C

DEFINING A FUNCTION

A function in Python is a reusable block of code that performs a specified purpose. It begins with the keyword **'def'**, followed by the function name and parentheses enclosing optional parameters. Here's an example.

Example:

```
def greet(name):
   print("Hello, " + name + "!")
```

REASONS OF USING FUNCTIONS

- Functions improve code readability
- It also improve code's reusability and maintenance
- They encapsulate certain activities (which makes code more modular and manageable)

TYPES OF FUNCTIONS IN PYTHON

In Python, functions may be classified into different categories based on their use and structure:

Built-In Functions:
These are Python standard library functions that can be used without explicit specification. Examples are **print(), len(), max(), and min().**

User-defined Functions:
These are user-created functions that execute certain tasks. Users can define their functions by using the **'def'** keyword followed by the function name, parameters, and body.

Example:

```
def greet(name):
   print("Hello, " + name + "!")
```

Anonymous functions (also known as lambda functions):
Lambda functions are tiny, anonymous functions created with the lambda keyword. They are excellent for short-term tasks that need a basic function.

Example:

```
double = lambda x: x * 2
print(double(5))  # Output: 10
```

Recursive functions:
Recursive functions solve problems by calling themselves, either directly or indirectly. They are beneficial for projects that can be separated into smaller sub-problems.

Example:

```
def factorial(n):
   if n == 0:
      return 1
   else:
      return n * factorial(n-1)
```

Higher Order Functions:
These functions either take other functions as parameters or return functions as output. Examples are **map(), filter(), and sort().**

Example:

```
def square(x):
   return x * x

numbers = [1, 2, 3, 4, 5]
squared_numbers = map(square, numbers)
```

ADVANTAGES OF USER-DEFINED FUNCTION

- User-defined functions enable developers to encapsulate functionality
- Increase code reuse
- Improve readability
- They enable modular programming and abstraction

RULES IN DECLARING A FUNCTION IN PYTHON

- Function names must follow variable naming conventions
- Use lowercase letters and underscores to improve readability
- Function names should be descriptive but brief

PYTHON FUNCTION SYNTAX
The syntax for creating a function in Python has a specified structure:

```
def function_name(parameters):
    """
    Optional docstring explaining the purpose of the function.
    """
    # Function body - code block
    statements
    return value  # Optional return statement
```

def: It's a keyword that marks the beginning of a function definition.

function_name: This is the name assigned to the function, which follows the same rules as variable names.

Docstring: This is an optional multiline string that appears immediately following the function header. It is used for documentation, describing what the function performs.

Function body: This is the indented chunk of code that runs when the function is invoked. It comprises the statements necessary to complete the intended task.

Parameters: are optional. They are variables that store the arguments supplied to the function. Parameters are contained in parenthesis **"()"**. Commas separate several parameters.

Return statement: is optional, its purpose is to return a value from the function to the caller. If omitted, the function returns "**None"**.

FUNCTIONS, ARGUMENTS, & PARAMETER
In Python, functions can include parameters and arguments, which are essential for defining and invoking functions, respectively.

> **Parameters** are placeholders for the data that a function need to complete its task. They are listed within the parenthesis of the function definition. Parameters serve as variables in the function's scope.
>
> **Arguments** are the actual values that a function receives when it is called. They correspond to the parameters specified in the function. These values may be literals, variables, expressions, or even other function calls.

THE RETURN STATEMENT
In Python, the **'return'** statement is used to terminate a function while optionally returning a value to the caller. It terminates the function's execution and returns the provided value as the result of the function call.

```
def add(x, y):
    """
    This function adds two numbers and returns the result.
    """
    result = x + y
    return result

sum = add(3, 5)
print(sum)
```

- The return keyword is followed by the value (or expression) that the function should produce.
- If there is no return statement or the return statement does not include a value, the function returns None by default.
- When a return statement is executed, the function terminates immediately and any following code in the function is not executed.
