Last Lecture

• Create our own functions
  • Docstring
  • Location of functions in program
• Guideline
  • Generalization
• Why creating functions
Learning outcomes

At the end of this course, a student is expected to:

• Create (design) small to medium size programs using Python:
  • Decompose a Python program into **functions**
• Use the core features of Python to design programs to solve problems: variables, expressions, terminal input and output, type conversion, conditionals, iteration, **functions**, standard library modules
• Design programs requiring approximately 100 lines and 6 **functions** (of well-designed code)
• Describe the benefits of using **functions**
• Construct **functions** such that:
  • Functions have a single purpose (decomposition)
  • Functions are reusable (generalisation)
  • Functions include parameters and local variables
  • Functions return values
• etc...
Today’s Menu

• Execution flow - so far
  • Stack frame

• Execution flow through main part of program and functions
  • When functions are called:
    • Arguments and parameters matching
    • Value returned from functions to “caller”
  • Scope of local variable and parameters
  • Immutable versus mutable
Program execution – so far

What happens when a program is executed (when a Python program is interpreted)?

1. A stack frame is created in memory for the program’s execution
2. The execution flow starts at the top of the program and each Python statement is executed (interpreted) in turn
Function call execution

What happens when the execution flow reaches a function call?

1. The function is called -> the execution flows to the function definition
2. A stack frame is created in memory for the function’s execution
3. Argument(s) are matched with parameter(s) – order is important
4. The indented statements in function body are executed sequentially
Function call execution

What happens when the execution flow reaches the end of a function?

- The execution flow returns at the location from where it had originally called the function -> the **caller**
- If the execution flow returned from the function with a **value**, the execution flow uses this value to evaluate the rest of the expression/statement
What is a stack frame?

• Answer: A chunk of memory

• When a Python (main) program starts executing, a stack frame is allocated to its execution
  • The variables created in the program are allocated memory space inside this stack frame

• When the execution of Python (main) program terminates, its stack frame is recycled (its variables are no longer accessible)
What is a stack frame?

- Every time a function is called, a stack frame is allocated to its execution
  - The parameter(s) (if any), when created, are allocated memory space inside this stack frame
  - The variables, local to this function, when created, are allocated memory space inside this stack frame

- When the execution of the function terminates, its stack frame is recycled (its parameters and variables are no longer accessible)
Scope of a variable

• **Scope of a variable** refers to the section of a program over which a variable is known and can be used
  
  • A variable is known from the moment it is created to the end of its scope
Scope: Local variables and parameters

- **Local variables and parameters**: variables known “locally” within a function, i.e., local to a function
  - A variable is known from the moment it is created to the end of its scope
  - A parameter is known from the moment the function is called to the end of the function’s execution (parameter’s scope)
Scope: Global variable

- **Global variable**: variable known over the entire Python program (the main section and all the functions), from the moment it is created to the end of its scope
Demo – Effect of mutability

- What happens when we call a function with argument(s) of mutable / immutable data types?
Summary

• Execution flow - so far
  • Stack frame

• Execution flow through main part of program and functions
  • When functions are called:
    • Arguments and parameters matching
    • Value returned from functions to “caller”
  • Scope of local variable and parameters
  • Immutable versus mutable
Next Lecture

• Module and Python Turtle
• Case study: creating a program with functions