CMPT 120

Topic: Functions – Part 4
Developing Software that incorporates Functions
Last Lecture

• Execution flow – so far, i.e., program with no user-defined functions
  • 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
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

• Case study: developing software that incorporates functions
• In the process, we shall point out a few guidelines:
  • Decomposition
  • Incremental Development
  • Function Interface Design
  • Generalization
  • Composition
  • Encapsulation
Creating functions in our software

Two ways of going about this!

**Way 1**
- If the software does not already exist, we can design and implement our solution incorporating functions

**Way 2**
- If the software already exist, we can encapsulate some of its code fragments (the ones with one specific purpose/repeated code fragments) into functions
Way 1: Developing software incorporating functions

- Incorporating functions into our software as we are developing it!

Let’s illustrate the development of software (a Python program) incorporating functions with a case study called **Area Calculator**
Step 1 – Problem Statement

- **Problem statement:** Develop a Python program to compute the area of various shapes: triangle, circle, rectangle, square, ellipse
Step 2 – Applying Decomposition

As we design a solution, we decompose it into actions --> functions

1. Print menu
2. Get option from user + validate
3. Get input from user + validate
4. Compute desired area
5. Display result

Compute the area of various shapes: triangle, circle, rectangle, square, ellipse
Decomposition in the real world

Getting myself to SFU in the morning

- Wake up
- Wash up
- Get dressed
- Eat breakfast
- Take the bus
Step 2 – Algorithm

- Each action -> a step of the algorithm
- So, each of these steps has a purpose

- Note that this algorithm is not very detailed
  -> High-level algorithm

# Print menu
# Get option from user (+ validate input)
# Based on option selected by user,
# get appropriate input from user
# ( + validate input)
# Compute desired area
# Display result

So, each step could potentially be implemented as a function
For example …

# Print menu
-> became the function `printMenu()`

# Get option from user (+ validate input)
-> became the function `getSelection()`
Step 2 – Low-level Algorithm

# Print menu
  # Print description of program
  # Print menu displaying selection of shapes

# Get options from user (+ validate input)
  # Print input instruction to user and read user input
  # Validate input

# Based on option selected by user, get appropriate input from user (+ validate input)
  # If "triangle" is selected, then ask for the base and height
  # If "circle" is selected, then ask for the radius
  # if "rectangle" is selected, then ask for the width and height
  # if "square" is selected, then ask for one side
  # if "ellipse" is selected, then ask for both radii
  # Validate input

# Compute desired area
  # If "triangle" is selected, then compute area = 0.5 ( base * height )
  # If "circle" is selected, then compute area = pi * radius squared
  # if "rectangle" is selected, then compute area = width * height
  # if "square" is selected, then compute area = side squared
  # if "ellipse" is selected, then compute area = pi * radius1 * radius2

# Display result
  # Print the shape, the input data and the area
Step 4 - Implementation

- See Area Calculator program posted on our course web site
Versions to our Case Study - 1

- **AreaCalculator - version 1**: Demonstrating **incremental development** guideline by implementing and testing the first two steps of our algorithm.
- **AreaCalculator - version 2**: Demonstrating **incremental development** guideline by implementing the sections of our algorithm dealing with the rectangle.
- **AreaCalculator - version 3**: Demonstrating **incremental development** guideline by implementing the sections of our algorithm dealing with the square.
Versions to our Case Study - 2

- **AreaCalculator - version 4**: Demonstrating refactoring repeated code from the functions `square()` and `rectangle()` and encapsulating this repeated code into their own function:
  - `getUserInput( whichData, shape )` - called from `square()` and `rectangle()` to get and validate side, width or height from user
  - `areaOfParallelogram( base, height )` - called from `square()` and `rectangle()` to compute their area since square and rectangle are both parallelograms and therefore use the same area equation
  - `displayResult( theShape, area )` - called from the main part of the program to display the result since all shapes will have a resulting area to display
Versions to our Case Study - 3

• **Note**: Throughout the 4 versions of our AreaCalculator, we demonstrate how to design the interface of a function
  • Function’s purpose and name
  • Function’s parameter(s)
  • Function’s returned value
AreaCalculator – Main Loop

```python
notASelection = 'X'
...
# Main part of the program - top level (of execution)
...
# As long as the user enters a valid selection ...
while selectedShape != notASelection:
    area = 0
    # If "triangle" is selected?
    if selectedShape == "T":
        # deal with triangle
    # If "circle" is selected?
    elif selectedShape == "C":
        # deal with circle
    # If "rectangle" is selected?
    elif selectedShape == "R":
        theShape = "rectangle"
        area = rectangle()
    # If "square" is selected?
    elif selectedShape == "S":
        theShape = "square"
        area = square()
    # If "ellipse" is selected?
    elif selectedShape == "E" :
        # deal with ellipse
    ...
print("---")
```

Event loop

- printMenu() called
- getSelection(...) called
- displayResult(...) called
Way 2: Enhancing software by incorporating functions

- If the software already exist, we can encapsulate (i.e., refactor) some of its code fragments into functions using the following guidelines:
  - If a code fragment is made of logically related statements, i.e., the code fragment has **one well defined purpose**, put the code into a function and replace the code fragment in the main part of the program by a call to this function
  - If a code fragment is **repeated** in several places in the program, put the repeated code into a function and replace each instance of the repeated code in the main part of the program by a call to this function
Summary

- Developing Software that incorporates Functions
  - Way 1 – the program does not exist yet
  - Way 2 – the program has already been written
Next Lecture

• Recursion