CSc 110, Autumn 2016

Lecture 2: Functions
Review

• What is the output of the following `print` statements?

```python
print("this class\tis' the \"best\"")
```

• Write a `print` statement to produce this output:

```
/ \ // \ \ /// \ \ 
```
Comments

• **comment**: A note written in source code by the programmer to describe or clarify the code.
  • Comments are not executed when your program runs.

• Syntax:
  
  ```
  #  comment text
  ```

• Examples:
  
  ```
  # This is a one-line comment.
  # This is a very long
  # multi-line comment.
  ```
Comments example

# Suzy Student,
# CSc 110, Fall 2019
# Displays lyrics

# first line
print("When I first got into magic")
print("it was an underground phenomenon")
print()

# second line
print("Now everybody's like")
print("pick a card, any card")
functions
Algorithms

• **algorithm**: A list of steps for solving a problem.

• Example algorithm: "Bake sugar cookies"
  • Mix the dry ingredients.
  • Cream the butter and sugar.
  • Beat in the eggs.
  • Stir in the dry ingredients.
  • Set the oven temperature.
  • Set the timer for 10 minutes.
  • Place the cookies into the oven.
  • Allow the cookies to bake.
  • Spread frosting and sprinkles onto the cookies.
  • ...
Problems with algorithms

• **lack of structure**: Many steps; tough to follow.

• **redundancy**: Consider making a double batch...
  • Mix the dry ingredients.
  • Cream the butter and sugar.
  • Beat in the eggs.
  • Stir in the dry ingredients.
  • Set the oven temperature.
  • Set the timer for 10 minutes.
  • Place the first batch of cookies into the oven.
  • Allow the cookies to bake.
  • Set the timer for 10 minutes.
  • Place the second batch of cookies into the oven.
  • Allow the cookies to bake.
  • Mix ingredients for frosting.
  • ...

...
Structured algorithms

• **structured algorithm**: Split into coherent tasks.
  
  **1 Make the batter.**
  • Mix the dry ingredients.
  • Cream the butter and sugar.
  • Beat in the eggs.
  • Stir in the dry ingredients.

  **2 Bake the cookies.**
  • Set the oven temperature.
  • Set the timer for 10 minutes.
  • Place the cookies into the oven.
  • Allow the cookies to bake.

  **3 Decorate the cookies.**
  • Mix the ingredients for the frosting.
  • Spread frosting and sprinkles onto the cookies.
  ...
Removing redundancy

• A well-structured algorithm can describe repeated tasks with less redundancy.

1 Make the cookie batter.
• Mix the dry ingredients.
• ...

2a Bake the cookies (first batch).
• Set the oven temperature.
• Set the timer for 10 minutes.
• ...

2b Bake the cookies (second batch).
• Repeat Step 2a

3 Decorate the cookies.
• ...
functions

• **function**: A named group of statements.
  - denotes the *structure* of a program
  - eliminates *redundancy* by code reuse

• **procedural decomposition**:
  dividing a problem into functions

• Writing a function is like adding a new command to Python.
Declaring a function

*Gives your function a name so it can be executed*

- **Syntax:**

  ```python
def name() :
    statement
    statement
    ...
    statement
  ```

- **Example:**

  ```python
def print_warning():
    print("This product causes cancer")
    print("in lab rats and humans.")
  ```
Calling a function

Executes the function’s code

- Syntax:
  ```
  name()
  ```
  - You can call the same function many times if you like.

- Example:
  ```
  print_warning()  #separate multiple words with underscores
  ```

- Output:
  ```
  This product causes cancer in lab rats and humans.
  ```
Functions calling functions

def message1():
    print("This is message1.")

def message2():
    print("This is message2.")
        message1()
    print("Done with message2.")

message1()
message2()
print("Done with everything.")

• Output:
  This is message1.
  This is message2.
  This is message1.
  Done with message2.
  Done with main.
Control flow

• When a function is called, the program's execution...
  • "jumps" into that function, executing its statements, then
  • "jumps" back to the point where the function was called.
Structure of a program

• No code should be placed outside a function. Instead use a `main` function.

  • The one exception is a call to your main function

```python
def main():
    message1()
    message2()
    print("Done with everything.")

def message1():
    print("This is message1.")

def message2():
    print("This is message2.")
    message1()
    print("Done with message2.")
main()
```
When to use functions (besides main)

• Place statements into a function if:
  • The statements are related structurally, and/or
  • The statements are repeated.

• You should not create functions for:
  • An individual print statement.
  • Only blank lines.
  • Unrelated or weakly related statements.
    (Consider splitting them into two smaller functions.)
Functions question

• Write a program to print these figures using functions.
First version (unstructured):

- Create an empty program.
- Copy the expected output into it, surrounding each line with `print` syntax.
- Run it to verify the output.
Program version 1

def main():
    print("  ______")
    print(" / \\
    \ \\
    /")
    print(" \ \ / \\
    \ \\
    /")
    print(" ______/ \\
    \ \\
    /")
    print(
    print(" \______/ \\
    \ \\
    /")
    print(" /--------+")
    print() 
    print(" ______")
    print(" / \\
    \ \\
    /")
    print(" / STOP | \\
    \ \\
    /")
    print(" \______/ \\
    \ \\
    /")
    print()
    print(" ______")
    print(" / \\
    \ \\
    /")
    print(" /--------+")
    print(" +---------+")

main()
Development strategy 2

Second version (structured, with redundancy):

- Identify the structure of the output.
- Divide the code into functions based on this structure.
Output structure

The structure of the output:
- initial "egg" figure
- second "teacup" figure
- third "stop sign" figure
- fourth "hat" figure

This structure can be represented by functions:
- egg
- tea_cup
- stop_sign
- hat
def main():
    egg()
    tea_cup()
    stop_sign()
    hat()

def egg():
    print("  ______")
    print("  /     \")
    print(" /       ")
    print(" \       /")
    print(" \_______/")
    print()

def tea_cup():
    print("  ____")
    print(" /     ")
    print(" /       ")
    print(" \       /")
    print(" \_______/")
    print("+--------+")
    print()

def stop_sign():
    print("  ______")
    print(" /     ")
    print(" /       ")
    print(" | STOP  |")
    print(" \       /")
    print(" \_______/")
    print()

def hat():
    print("  ______")
    print(" /     ")
    print(" /       ")
    print(" +-------+")
    print()
Development strategy 3

Third version (structured, without redundancy):

- Identify redundancy in the output, and create functions to eliminate as much as possible.
- Add comments to the program.
Output redundancy

The redundancy in the output:

- egg top: reused on stop sign, hat
- egg bottom: reused on teacup, stop sign
- divider line: used on teacup, hat

This redundancy can be fixed by functions:

- egg_top
- egg_bottom
- line
Program version 3

# Suzy Student, CSc 110, Spring 2094
# Prints several figures, with methods for structure and redundancy.
def main():
    egg()
    tea_cup()
    stop_sign()
    hat()

# Draws the top half of an egg figure.
def egg_top():
    print("    _____")
    print("    /    \")
    print("    /      ")

# Draws the bottom half of an egg figure.
def egg_bottom():
    print("      /")
    print("     \____/")

# Draws a complete egg figure.
def egg():
    egg_top()
    egg_bottom()
    print()

# Draws a teacup figure.
def tea_cup():
    egg_bottom()
    line()
    print()

# Draws a stop sign figure.
def stop_sign():
    egg_top()
    print("| STOP |")
    egg_bottom()
    print()

# Draws a figure that looks sort of like a hat.
def hat():
    egg_top()
    line()

# Draws a line of dashes.
def line():
    print("+--------+")