CSc 110, Spring 2017

Lecture 2: Functions

Adapted from slides by Marty Stepp and Stuart Reges

webs
IMPORT OS
CUR_PATH = OS.GETCWD()
IGNORE_SET = SET(["__INIT__.PY", "COUNT_SOURCELINES.PY"])
YEAH, HE'S A PARSER-TONGUE.
I DIDN'T KNOW HARRY SPOKE PYTHON.

Review

- From last lecture: print, strings, escape sequences.
- What is the output of the following statement?

```
print("Who said,\"To thine own self be true.\"?")
```

- 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

Python statement              # comment text
```

- Examples:

```
# This is a one-line comment.

# This is a very long
# multi-line comment.
print("Hello!")                 # Output a greeting
```

Comments example

```
# Suzy Student
# CSc 110
# Displays lyrics

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

# second part
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.
    - Mix ingredients for frosting.
    - Spread frosting and sprinkles onto the cookies.

Problems with this algorithm

- *lack of structure*: Many steps; tough to follow at a glance.
- What if there is batter for 24 cookies and the baking sheet fits only 12?
  - 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 oven temperature.
  - Set the timer for 10 minutes.
  - Place the second batch of cookies into the oven.
  - Allow the cookies to bake.
  - Mix ingredients for frosting.
  - Spread frosting and sprinkles on the cookies
  - *Repetition*: Steps are listed twice

Structured algorithms

- **structured algorithm**: Decomposed into related 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 repetition

- A well-structured algorithm can describe repeated steps easily.

  1 Make the 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.
  - Mix the ingredients for the frosting.
  - ...

functions

- **function**: A named sequence of statements.
    - supports the creation of *well-structured* programs
    - eliminates *repetition* by code reuse

- **procedural decomposition**:
  dividing a problem or sequence of statements into functions

- Example:

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

Declaring a function

Gives your function a name so it can be executed

- Syntax:

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

Space is part of the syntax.

Statements in a function must have the same level of indentation.

- Function names:
  Consist of upper and lower case letters, "_", and digits 0 through 9.

- Example:

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

Calling a function

Executes the statements within a function

- Syntax:

  name ()

  - You can call the same function many times if you like.

- Example:

```
print_warning()          #using underscores makes names readable
```

  - Output:

```
This product causes cancer
in lab rats and humans.
```

Functions calling functions

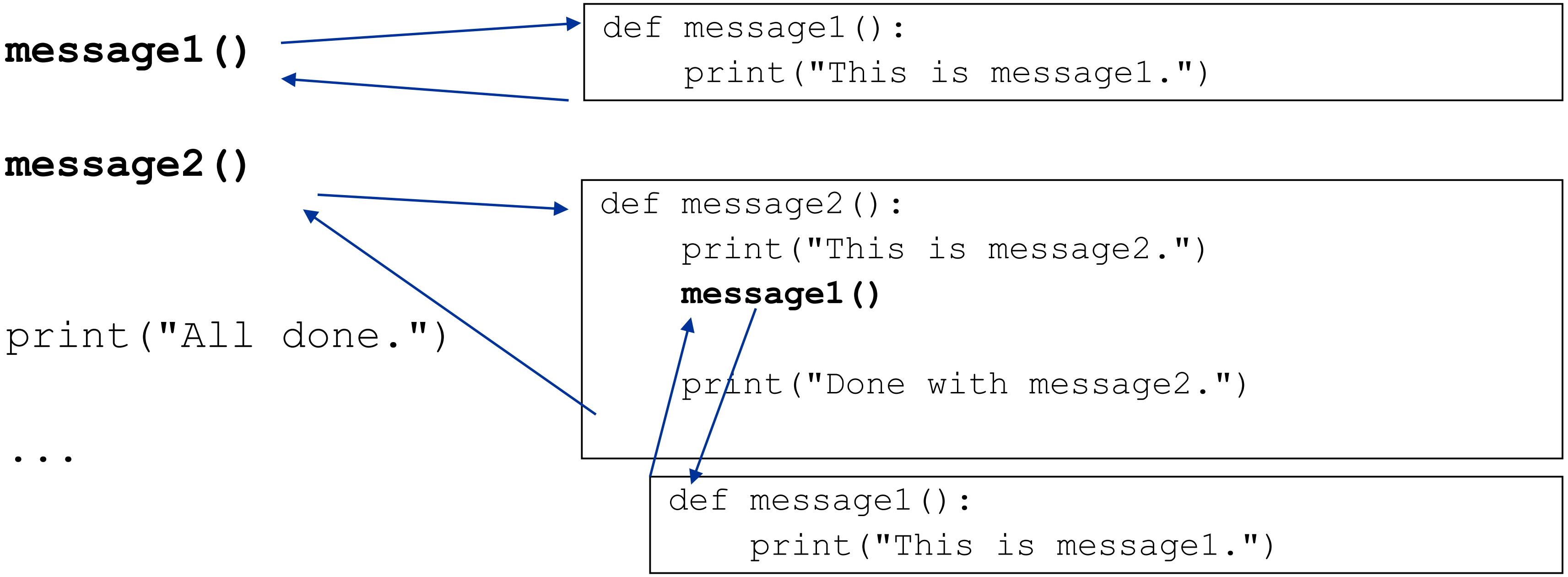
```
def message1():
    print("This is message1.")

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

message1()
message2()
print("All done.")
```

- Output:

```
This is message1.
This is message2.
This is message1.
Done with message2.
All done.
```

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.



```
message1()

message2()


print("All done.")

...
```

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

```
def message2():
    print("This is message2.")
    message1()

    print("Done with message2.")
```

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

Structure of a program

- Best practice for well-structured programs: all code should be placed inside a function.

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

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

message1()
message2()
print("Done with all.")
```

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

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

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

main()
```

Use a function called `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.
  - Unrelated or weakly related statements.
    (Consider splitting them into two smaller functions.)

Problem

- Write a program to print these figures using functions.

```
  ______
 /      \
/        \
\        /
 \______/

\        /
 \______/
+--------+

  ______
 /      \
/        \
|  STOP  |
\        /
 \______/

  ______
 /      \
/        \
+--------+
```

Development strategy

```
  ______
 /      \
/        \
\        /
 \______/

\        /
 \______/
+--------+

  ______
 /      \
/        \
|  STOP  |
\        /
 \______/

  ______
 /      \
/        \
+--------+
```

Approach – simply get this to print correctly

First version (unstructured):

- Start IDLE. File->open -> new
- 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("  ______")
    print(" /      \\")
    print("/        \\")
    print("|  STOP  |")
    print("\\        /")
    print(" \\______/")
    print()
    print("  ______")
    print(" /      \\")
    print("/        \\")
    print("+--------+")

main()
```

Development strategy 2

```
  ______
 /      \
/        \
\        /
 \______/

\        /
 \______/
+--------+

  ______
 /      \
/        \
|  STOP  |
\        /
 \______/

  ______
 /      \
/        \
+--------+
```

Second version (structured, with repetition):

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

Output structure

```
  ______
 /      \
/        \
\        /
 \______/
```

```
\        /
 \______/
+--------+
```

```
  ______
 /      \
/        \
|  STOP  |
\        /
 \______/
```

```
  ______
 /      \
/        \
+--------+
```

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`

Program version 2

```
def main():
    egg()
    tea_cup()
    stop_sign()
    hat()

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

def tea_cup():
    print("\\        /")
    print(" \\______/")
    print("+--------+")
    print()
```

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

def hat():
    print("  ______")
    print(" /      \\")
    print("/        \\")
    print("+--------+")
```

Development strategy 3

```
  ______
 /      \
/        \
\        /
 \______/

\        /
 \______/
+--------+

  ______
 /      \
/        \
|  STOP  |
\        /
 \______/

  ______
 /      \
/        \
+--------+
```

Third version (structured, without repetition):

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

Repetition in the output

```
  ______
 /      \
/        \
\        /
 \______/

\        /
 \______/
+--------+

  ______
 /      \
/        \
|  STOP  |
\        /
 \______/

  ______
 /      \
/        \
+--------+
```

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 functions for structure and redundancy.
def main():
    egg()
    tea_cup()
    stop_sign()
    hat()

# Draws the top half of an 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():
    eggTop()
    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("+--------+")
```