Introduction to Problem Solving in Python

COSI 10A

Review

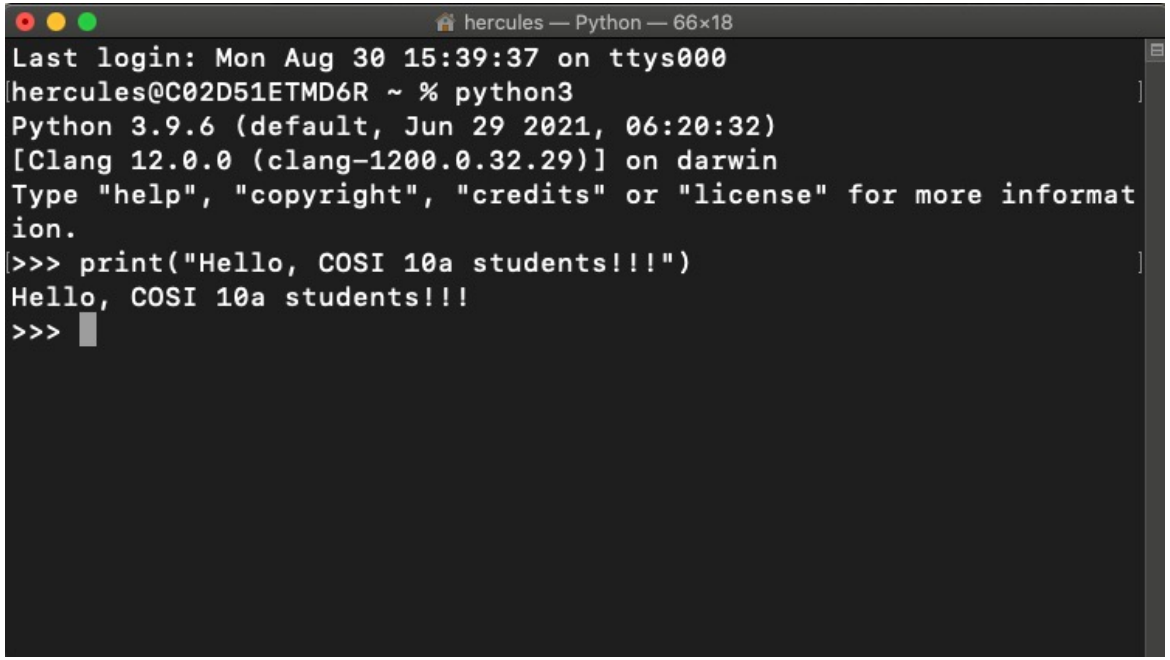
- Interpreter VS Compilers
- interactive mode VS script mode
- print() statement
- strings, and escape sequences

Review: Your first program

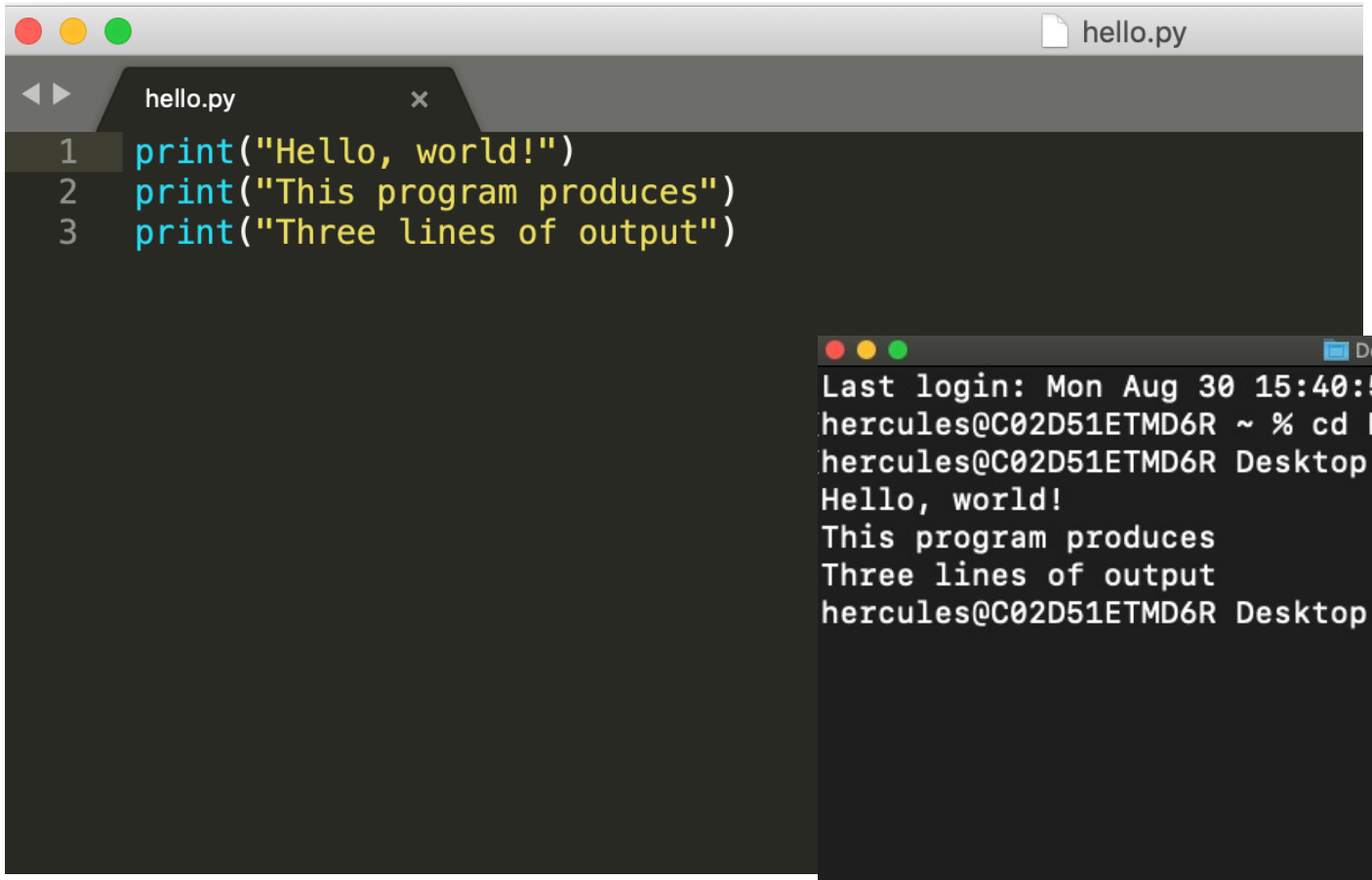
- *Print (or display) on the console Hello, COSI 10a students!!!*

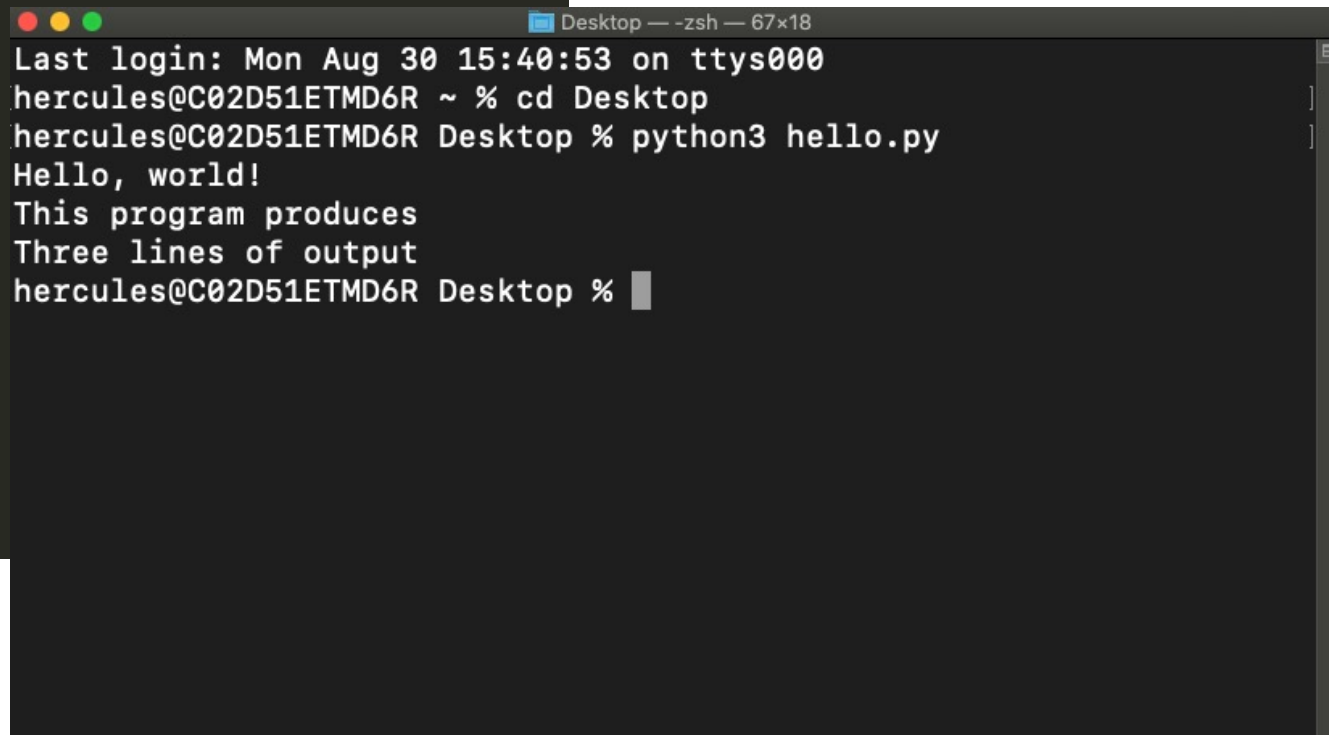
- The **console** is the text box into which the program's output is printed

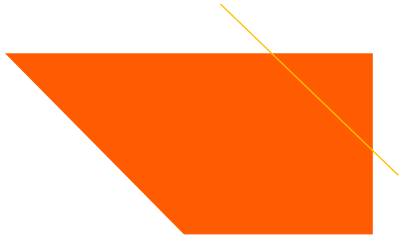
```
print("Hello, COSI 10a students!!!")
```



Review: Creating a Python Program File







Class objectives

- Understand procedural decomposition
- Use of functions

Comments

Comments

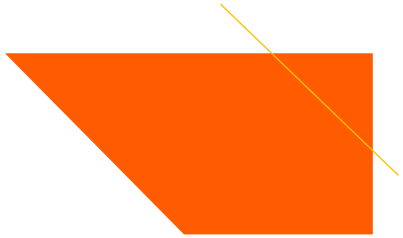
- A comment is a note written in the source code by the programmer to describe or clarify the code
- Comments are not executed when your program runs

- **Syntax**:

  `#` **comment text**

- **Examples**:

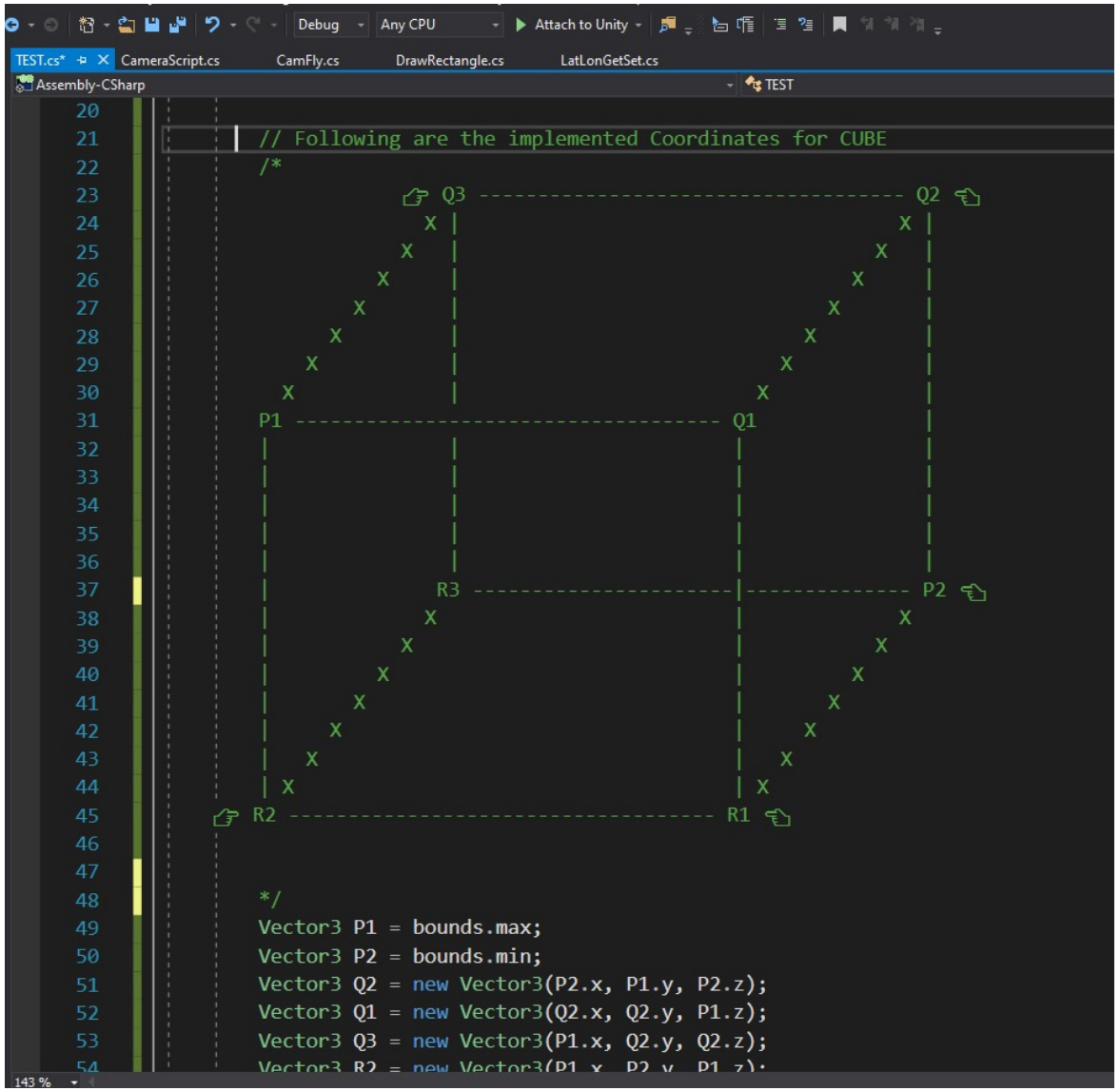
```
# This is a one-line comment.

# This is a very long
# multi-line comment.
```

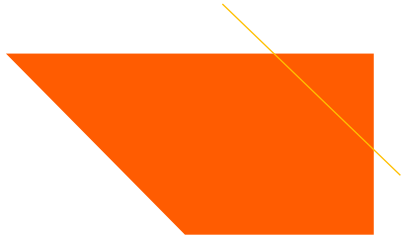


Comments

```
# Iraklis Tsekourakis
# CS 10a, Fall 2021
# My first Python program

# first line
print("Congratulations!")
print()
print("Today is your day")


print("""You're off to Great Places!
You're off and away!\n""")
```

Comments!!


Debug
Any CPU
Attach to Unity
TEST.cs*
CameraScript.cs
CamFly.cs
DrawRectangle.cs
LatLonGetSet.cs
Assembly-CSharp
TEST
// Following are the implemented Coordinates for CUBE
/*
Q3
Q2
P1
Q1
R3
P2
R2
R1
*/
Vector3 P1 = bounds.max;
Vector3 P2 = bounds.min;
Vector3 Q2 = new Vector3(P2.x, P1.y, P2.z);
Vector3 Q1 = new Vector3(Q2.x, Q2.y, P1.z);
Vector3 Q3 = new Vector3(P1.x, Q2.y, Q2.z);
143 %



Functions

Algorithm

- An **algorithm** is 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 the “bake cookies” algorithm

- **Lack of structure**: Many steps; tough to follow
  - 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
  - Mix ingredients for frosting
  - …

Problems with the “bake cookies” algorithm

- **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 “bake cookies” algorithms

- Split into coherent tasks:
  - **Make the batter**
    - Mix the dry ingredients
    - Cream the butter and sugar
    - Beat in the eggs
    - Stir in the dry ingredients
  - **Bake the cookies**
    - Set the oven temperature
    - Set the timer for 10 minutes
    - Place the cookies into the oven
    - Allow the cookies to bake
  - **Decorate the cookies**
    - Mix the ingredients for the frosting.
    - Spread frosting and sprinkles onto the cookies.
    - ...

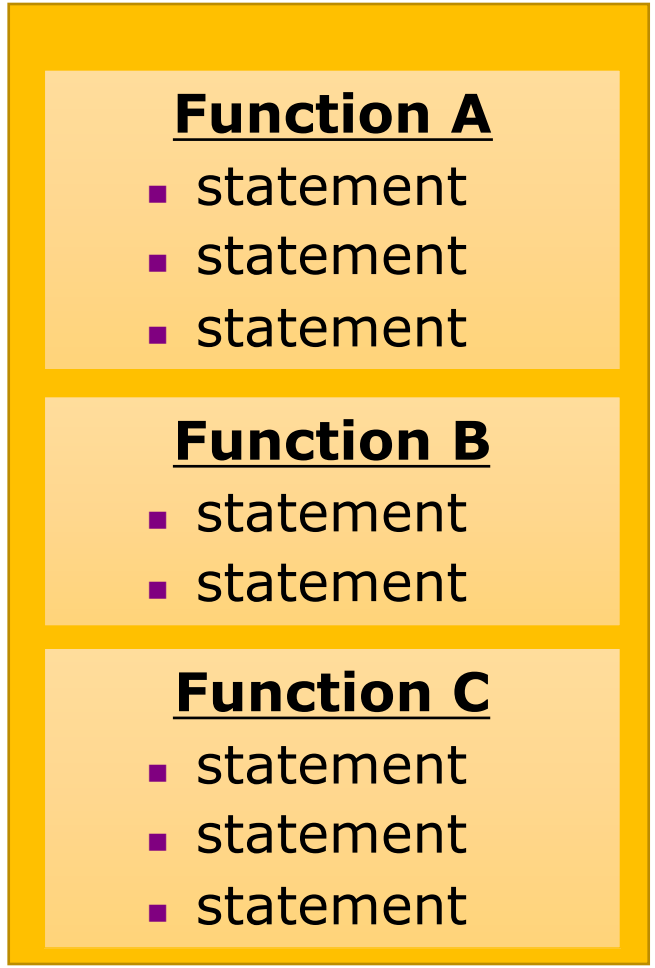
- Mix the dry ingredients
- Cream the butter and sugar
- Beat in the eggs
- Stir in the dry ingredients
- Set the oven temperature
- Set the timer for 10 minutes
- Place the first batch of cookies into the oven
- Allow the cookies to bake
- Set the timer for 10 minutes
- Place the second batch of cookies into the oven
- Allow the cookies to bake
- Mix ingredients for frosting
- ...

Removing redundancy

- A well-structured algorithm can describe repeated tasks with less redundancy
  - **Make the cookie batter**
    - Mix the dry ingredients.
    - ...
  - **Bake the cookies (first batch)**
    - Set the oven temperature.
    - Set the timer for 10 minutes.
    - ...
  - **Bake the cookies (second batch)**
    - Repeat Bake the cookies (first batch)
  - **Decorate the cookies**
    - ...

Functions

- A **function** is 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 functions

- To declare a function you must give a name to it, so later it can be executed

- Syntax:

```
def name():
    statement
    statement
    …
    statement
```

- Example:

NOTE: Separate multiple words with underscores

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

Calling a functions

- To call a function it means execute the function's statements
- Syntax: `name()`
- Example:

```
print_warning()
```

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

Program with functions

```
# This function prints the lyrics to my favorite song.
def rap():
    print("Now this is the story all about how")
    print("My life got flipped turned upside-down")

rap()                   # Calling (running) the rap function
print()
rap()                   # Calling the rap function again
```

```
Now this is the story all about how
My life got flipped turned upside-down

Now this is the story all about how
My life got flipped turned upside-down
```

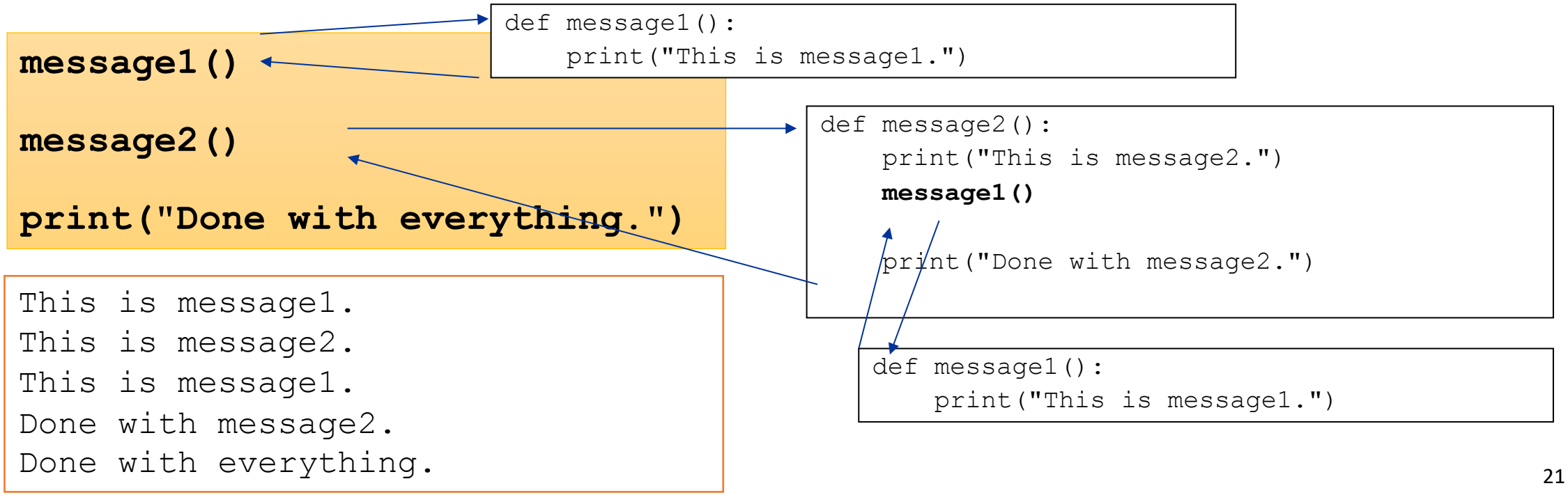
Functions calling functions

```
# This program shows functions calling other functions
def message1():
    print("This is message1.")


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

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

```
This is message1.
This is message2.
This is message1.
Done with message2.
Done with everything.
```

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

```
# This program shows functions calling other functions

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()
```

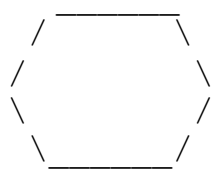
Call to the main function

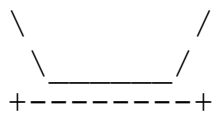
When to use functions (besides `main`)

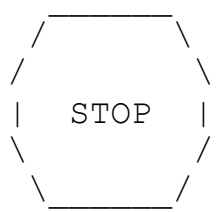
- Place statements into a function if:
  - The statements are related structurally, and/or
  - The statements are repeated
- You should not create functions for:
  - An individual print statement
  - Only blank lines
  - Unrelated or weakly related statements

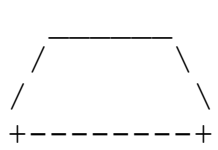
Functions question

- Write a program to print these figures using functions



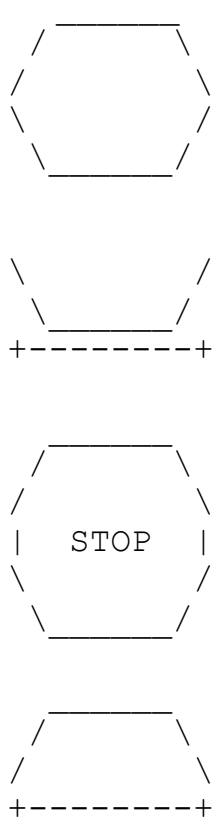






version 1

figurev1.py

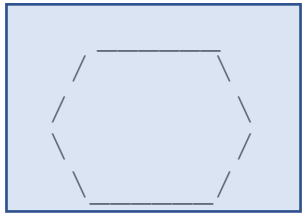



```
def main():
    print("   ______")
    print(" /       \\")
    print("/         \\")
    print("\\         /")
    print(" \\______/")
    print()
    print("\\         /")
    print(" \\______/")
    print("+--------+")
    print()
    print("   ______")
    print(" /       \\")
    print("/  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

main()
```

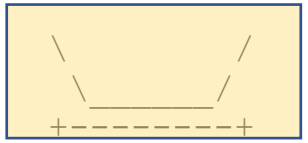
Unstructured

version 2

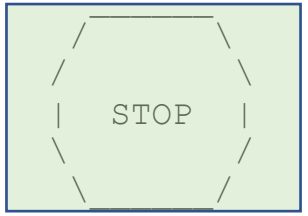
Divide the code into functions



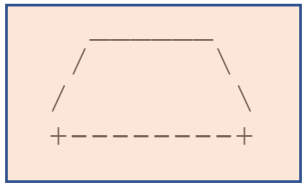
egg



tea_cup



stop_sign



hat

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

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

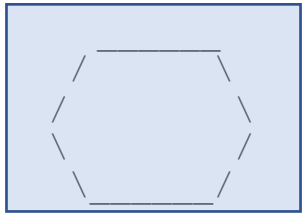
```
def stop_sign():
    print("  _______")
    print(" /       \\")
    print("/         \\")
    print("|  STOP   |")
    print("\\         /")
    print(" \\_______/")
    print()
```

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

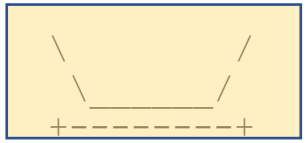
version 2

figurev2.py

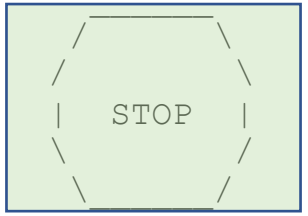
Divide the code into functions



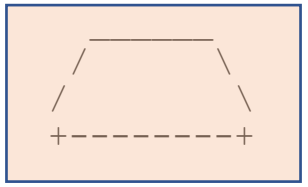
egg



tea_cup



stop_sign



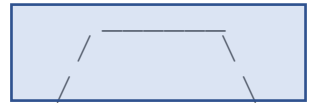
hat

Structured but with redundancy (or duplicate code)

```
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("+--------+")
main()
```

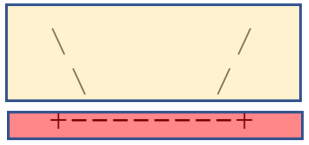
version 3

Divide the code into functions



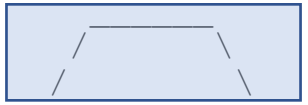
`egg_top`

`egg_bottom`

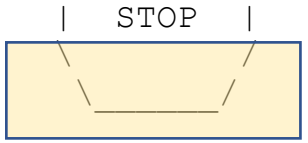


`egg_bottom`

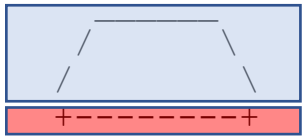
`line`



`egg_top`



`egg_bottom`



`egg_top`

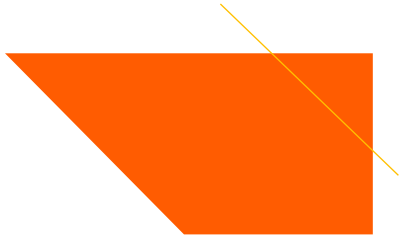
`line`

- `egg_top`: reused on stop sign, hat
- `egg_bottom`: reused on teacup, stop sign
- `line`: used on teacup, hat

version 3

```
# Prints several figures, with methods 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():
    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("+--------+")
main()
```



version 3

- The third version is structured and without redundancy
  - Identify redundancy in the output, and create functions to eliminate as much as possible
  - Add comments to the program

Why functions?

- Functions give you an opportunity to name a group of statements, which makes your program easier to read and debug
- Functions make a program smaller by eliminating repetitive code (or redundancy)
  - If you make a change you only have to make it in one place
- Dividing a long program into functions allows you to debug the parts one at a time and then assemble them into a working program
- Well-designed functions are often useful for many programs. Once you write and debug one, you can reuse it