

# Building Python Programs

## Chapter 1: Introduction to Python

# Computer Science

- CS is about PROCESS – describing how to accomplish tasks
  - "efficiently implementing automated abstractions" ([Philip Guo](#))
- Computers are a tool
  - Currently the best implementation platform
  - What kinds of problems can they solve?
  - How can they be made faster, cheaper, more efficient...?
- Science?
  - More like engineering, art, magic...
  - Hypothesis creation, testing, refinement important
- CS is still a young field finding itself

# Programming

- **program:** A set of instructions to be carried out by a computer.
- **program execution:** The act of carrying out the instructions contained in a program.
- **programming language:** A systematic set of rules used to describe computations in a format that is editable by humans.



# Some modern languages

- *procedural languages*: programs are a series of commands
  - **Pascal** (1970): designed for education
  - **C** (1972): low-level operating systems and device drivers
- *functional programming*: functions map inputs to outputs
  - **Lisp** (1958) / **Scheme** (1975), **ML** (1973), **Haskell** (1990)
- *object-oriented languages*: programs use interacting "objects"
  - **Smalltalk** (1980): first major object-oriented language
  - **C++** (1985): "object-oriented" improvements to C
    - successful in industry; used to build major OSes such as Windows
  - **Python** (1991):
    - The language taught in this course

# Why Python?

- Relatively simple
- Pre-written software
- Widely used

# A Python program

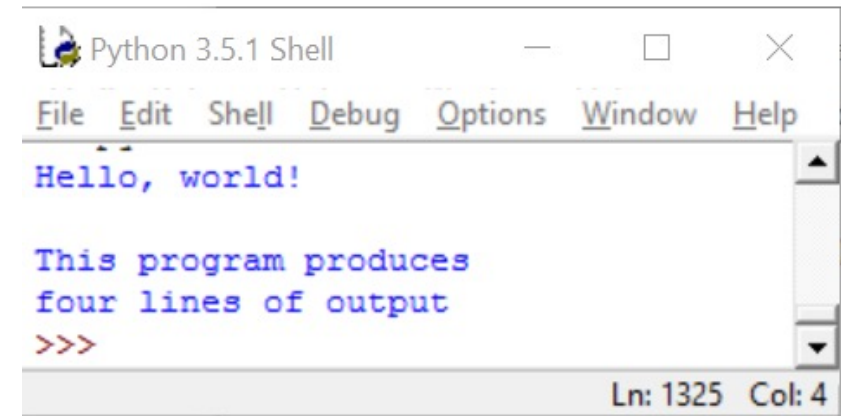
```
print("Hello, world!")  
print()  
print("This program produces")  
print("four lines of output")
```

- **Its output:**

Hello, world!

This program produces  
four lines of output

- **console:** Text box into which the program's output is printed.



# `print`

- A statement that prints a line of output on the console.
- Two ways to use `print` :
  - `print ("text")`  
Prints the given message as output.
  - `print ()`  
Prints a blank line of output.

# Strings and escape sequences



# Strings

- **string:** A sequence of characters to be printed.
  - Starts and ends with a " quote " character or a ' quote ' character.
    - The quotes do not appear in the output.
  - Examples:

```
"hello"  
"This is a string. It's very long!"  
'Here is "another" with quotes in'  
"""I can span multiple lines  
because I'm surrounded by 3 quotes"""
```
- Restrictions:
  - Strings surrounded by " " or ' ' may not span multiple lines

```
"This is not  
a legal String."
```
  - Strings surrounded by " " may not contain a " character.

```
"This is not a "legal" String either."
```
  - Strings surrounded by ' ' may not contain a ' character.

```
'This is not a 'legal' String either.'
```

# Escape sequences

- **escape sequence:** A special sequence of characters used to represent certain special characters in a string.

<code>\t</code>	tab character
<code>\n</code>	new line character
<code>\"</code>	quotation mark character
<code>\'</code>	quotation mark character
<code>\\</code>	backslash character

- **Example:**

```
print("\\hello\nhow\tare \"you\"?\\\\\\")
```

- **Output:**

```
\hello
how      are "you"?\\
```

# Questions

- What `print` statements will generate this output?

```
This quote is from  
Irish poet Oscar Wilde:
```

```
"Music makes one feel so romantic  
- at least it always gets on one's nerves -  
which is the same thing nowadays."
```

- What `print` statements will generate this output?

```
A "quoted" String is  
'much' better if you learn  
the rules of "escape sequences."
```

```
Also, "" represents an empty String.  
Don't forget: use \" instead of " !  
' is not the same as "
```

# Answers

- `print` statements to generate the output:

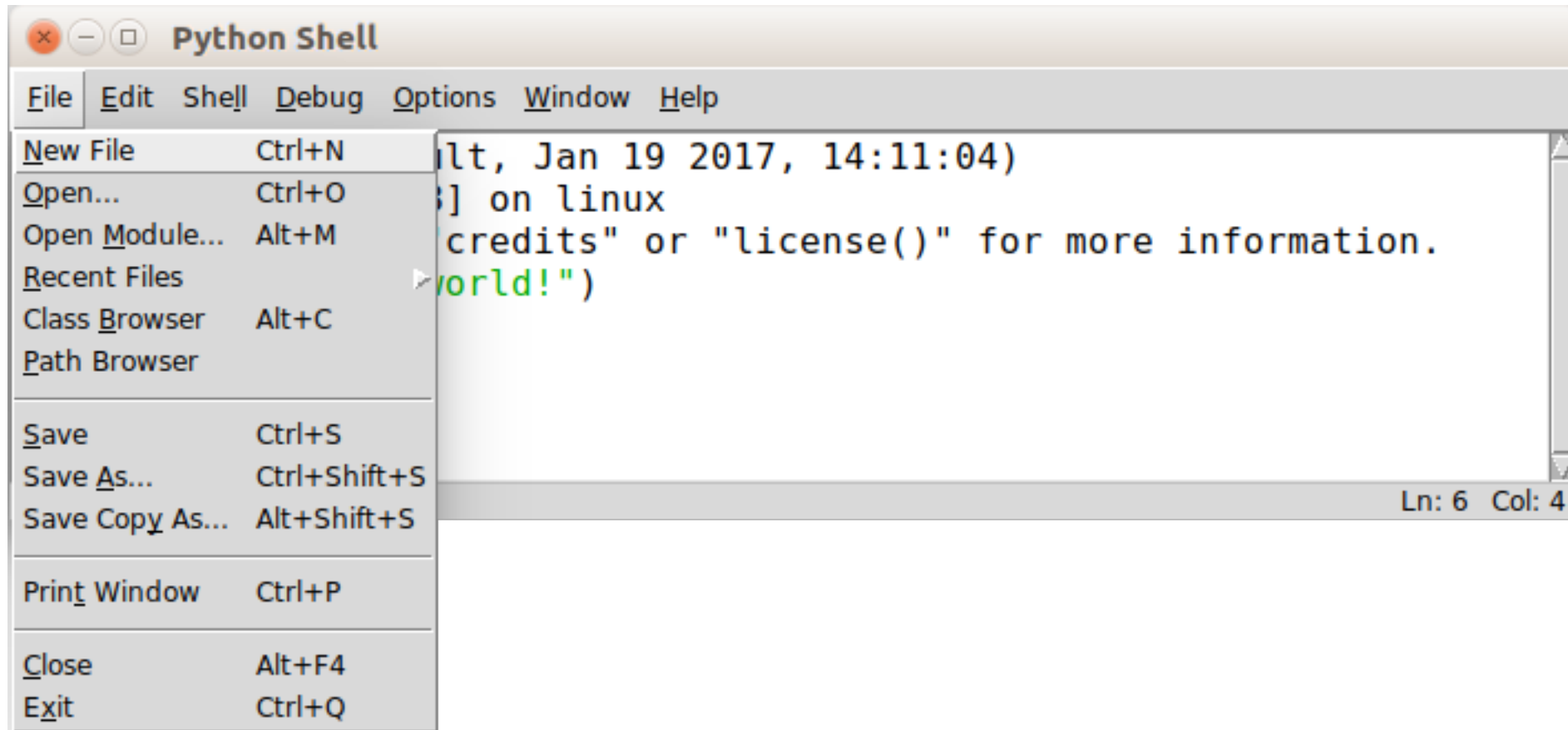
```
print("This quote is from")
print("Irish poet Oscar Wilde:")
print()
print("\"Music makes one feel so romantic")
print("- at least it always gets on one's nerves -")
print("which is the same thing nowadays.\"")
```

- `print` statements to generate the output:

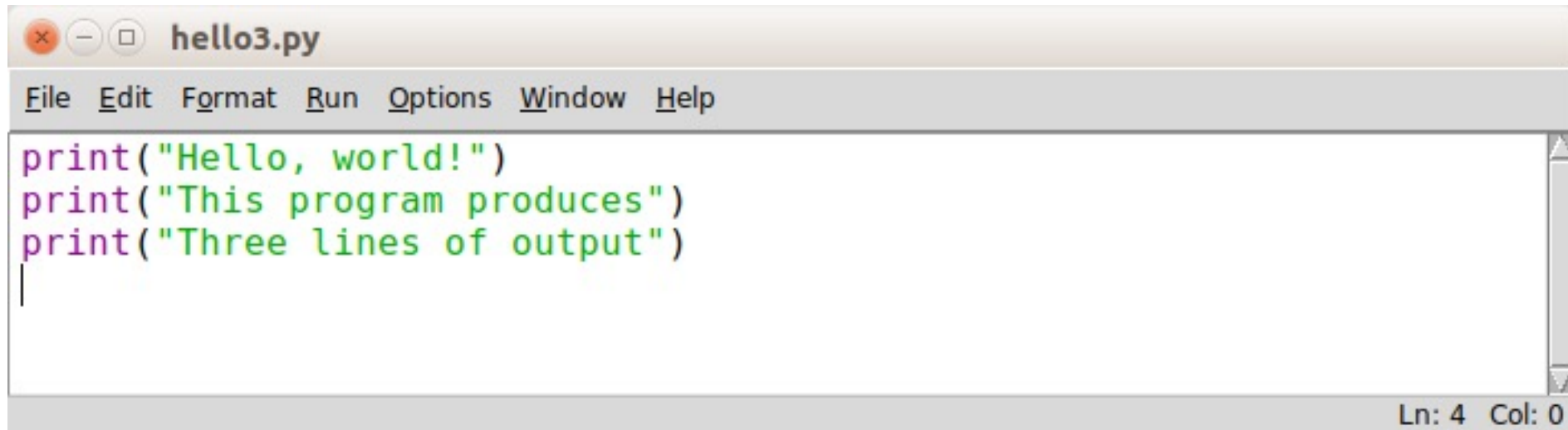
```
print("A \"quoted\" String is")
print("'much' better if you learn")
print("the rules of \"escape sequences.\"")
print()
print("Also, \"\" represents an empty String.")
print("Don't forget: use \"\" instead of \" !")
print("' ' is not the same as \"")
```

# Creating a Python Program

# Creating a Python Program File



# Creating a Python Program File

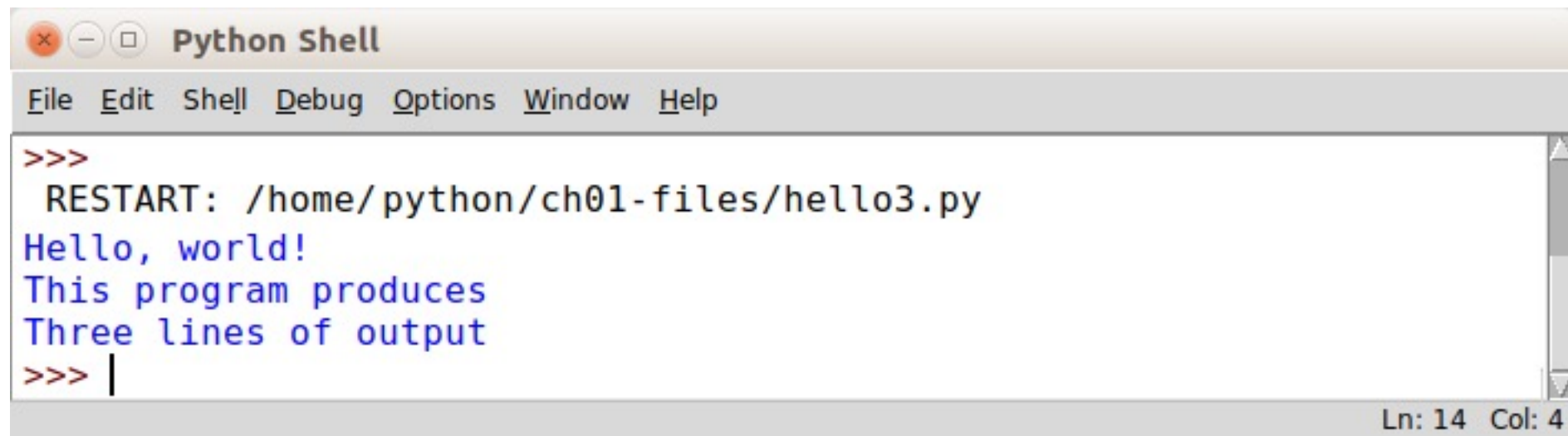


A screenshot of a Python IDE window titled "hello3.py". The window has a menu bar with "File", "Edit", "Format", "Run", "Options", "Window", and "Help". The main text area contains the following Python code:

```
print("Hello, world!")  
print("This program produces")  
print("Three lines of output")  
|
```

The status bar at the bottom right indicates "Ln: 4 Col: 0".

When Run -> Run Module is selected:



A screenshot of a Python Shell window titled "Python Shell". The window has a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The main text area shows the execution of the script:

```
>>>  
RESTART: /home/python/ch01-files/hello3.py  
Hello, world!  
This program produces  
Three lines of output  
>>> |
```

The status bar at the bottom right indicates "Ln: 14 Col: 4".

# 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).

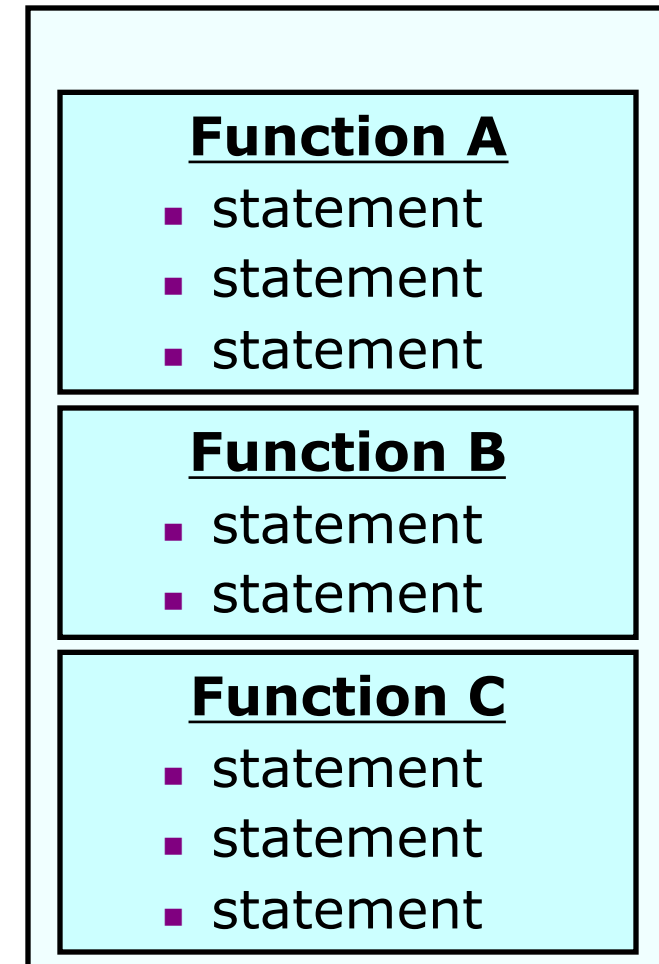
- Repeat Step 2a

## 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:

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

- Example:

```
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.
```

# Using functions

## 1. **Design** (think about) the algorithm.

- Look at the structure, and which commands are repeated.
- Decide what are the important overall tasks.

## 2. **Declare** (write down) the functions.

- Arrange statements into groups and give each group a name.

## 3. **Call** (run) the function.

# 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
```

## Output:

```
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

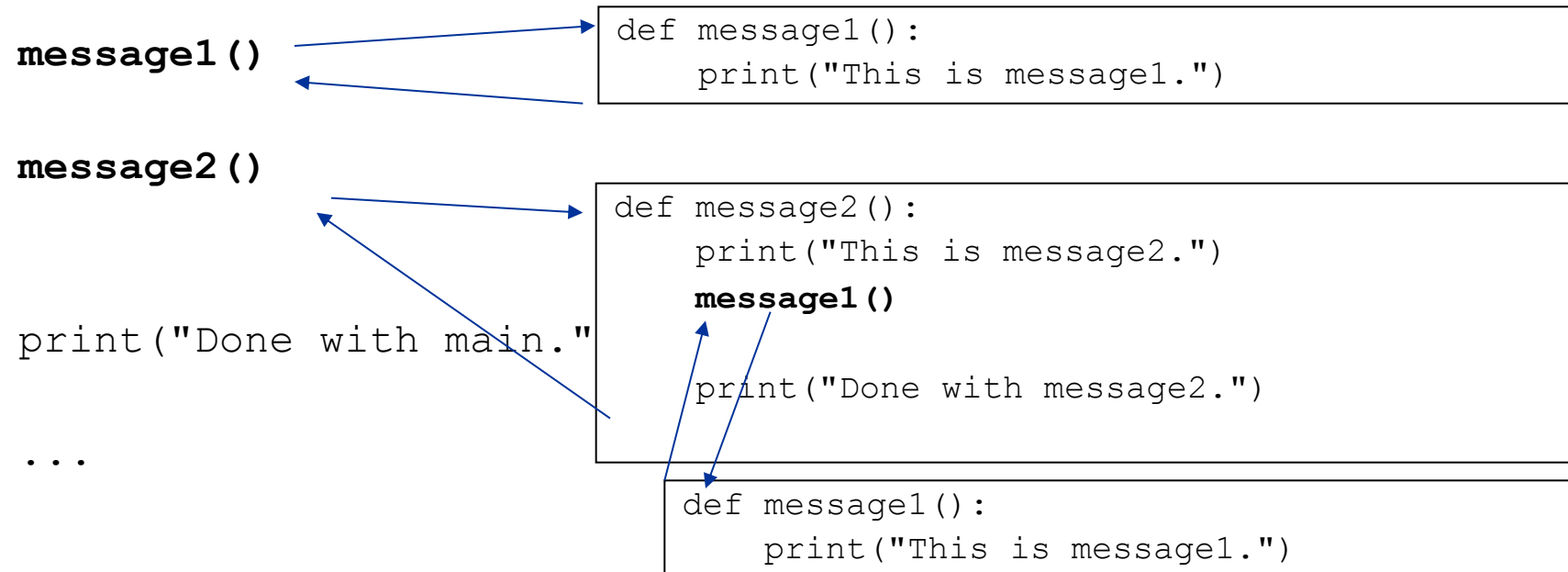
```
def message1():  
    print("This is message1.")  
  
def message2():  
    print("This is message2.")  
    message1()  
    print("Done with message2.")  
  
message1()  
message2()  
print("Done with everything.")
```

- **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

```
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.)

# Keywords

- **keyword:** An identifier that you cannot use because it already has a reserved meaning in Python.

and	del	from	not	while
as	elif	global	or	with
assert	else	if	pass	yield
break	except	import	print	
class	exec	in	raise	
continue	finally	is	return	
def	for	lambda	try	