Assigned: Quiz2
Vars and Exprs,
Using Functions,
Creating Functions,
Function Scope

[220 / 319] Function Scope

Meena Syamkumar
Andy Kuemmel
Cole Nelson

Readings:
Parts of Chapter 3 of Think Python

Due: P2
Grade posted: P1

Learning Objectives Today

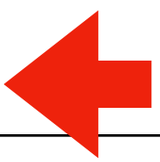
Explain rules of scope of **local variables** in a function

- When are they created?
- When are they destroyed?
- What parts of a program have access to them? (frames)

Understand **global variables**

- How can they be used and modified within a function?(global keyword)
- Where are they stored? (global frame)
- What parts of a program have access to them?
- How can they be mis-represented as local variables?

Read: Downey Ch 3 ("Parameters and Arguments" to end)
Link to Slides
Interactive Exercises



Explain **argument passing**

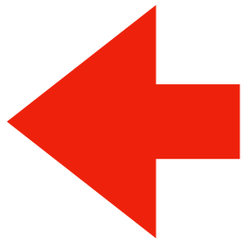
- "pass by value"

don't memorize the examples, learn the rules of Python

sample question: *why did Python do this thing I didn't expect at this specific line (ask us!)*

Today's Outline

Context
- Examples



Frames

Demos: Local Variables

Demos: Global Variables

Demos: Argument Passing

Context

Often (in life and programming), the same name can mean different things in different contexts

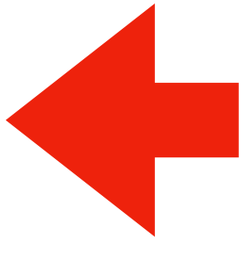
- Examples?
- Human name: Matthew (who is in the room?)
- Street address: 534 State Street (what city are we in?)
- Files: main.ipynb (which directory are we in?)

Our code often have different variables with the same name

- How do we keep variable names organized? with groups called "frames"
- How do we know what a variable name is referring to? we'll learn some rules for this

Today's Outline

Context

Frames 

Demos: Local Variables

Demos: Global Variables

Demos: Argument Passing

Frames

Every time a function is invoked (i.e., called), the invocation gets a new "frame" for holding variables

- The parameters also exist in a frame

Global frame

- There is always one global frame that all functions can access

When a variable name is used, Python looks two places:

1. the function invocation's frame
2. the global frame

Understanding scope: example

```
1  def print_twice(text):
2      print(text)
3      print(text)
4
5  def concatenate_str(text1, text2):
6      combined_text = text1 + text2 # concatenation
7      print_twice(combined_text)
8
9  cs220 = "Hello CS220"
10 cs319 = " / CS319"
11 concatenate_str(cs220, cs319)
```

two frames will exist during the time we're executing in print_twice

cs220 and cs319 will be in the global frame

you don't generally see or interact with frames when programming, but it's an important mental model

Understanding scope: example

```
1  def print_twice(text):
2      print(text)
3      print(text)
4
5  def concatenate_str(text1, text2):
6      combined_text = text1 + text2 # concatenation
7      print_twice(combined_text)
8
9  cs220 = "Hello CS220"
10 cs319 = " / CS319"
11 concatenate_str(cs220, cs319)
```

can access: `cs220, cs319, text`

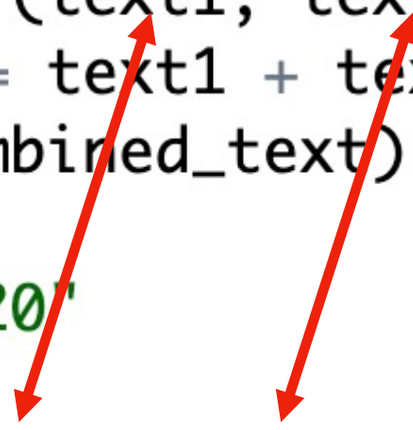
can access: `cs220, cs319, text1, text2, combined_text`

this code can access: `line1, line2`

we call the variables that can currently be accessed "in scope" and variables that cannot be "out of scope"

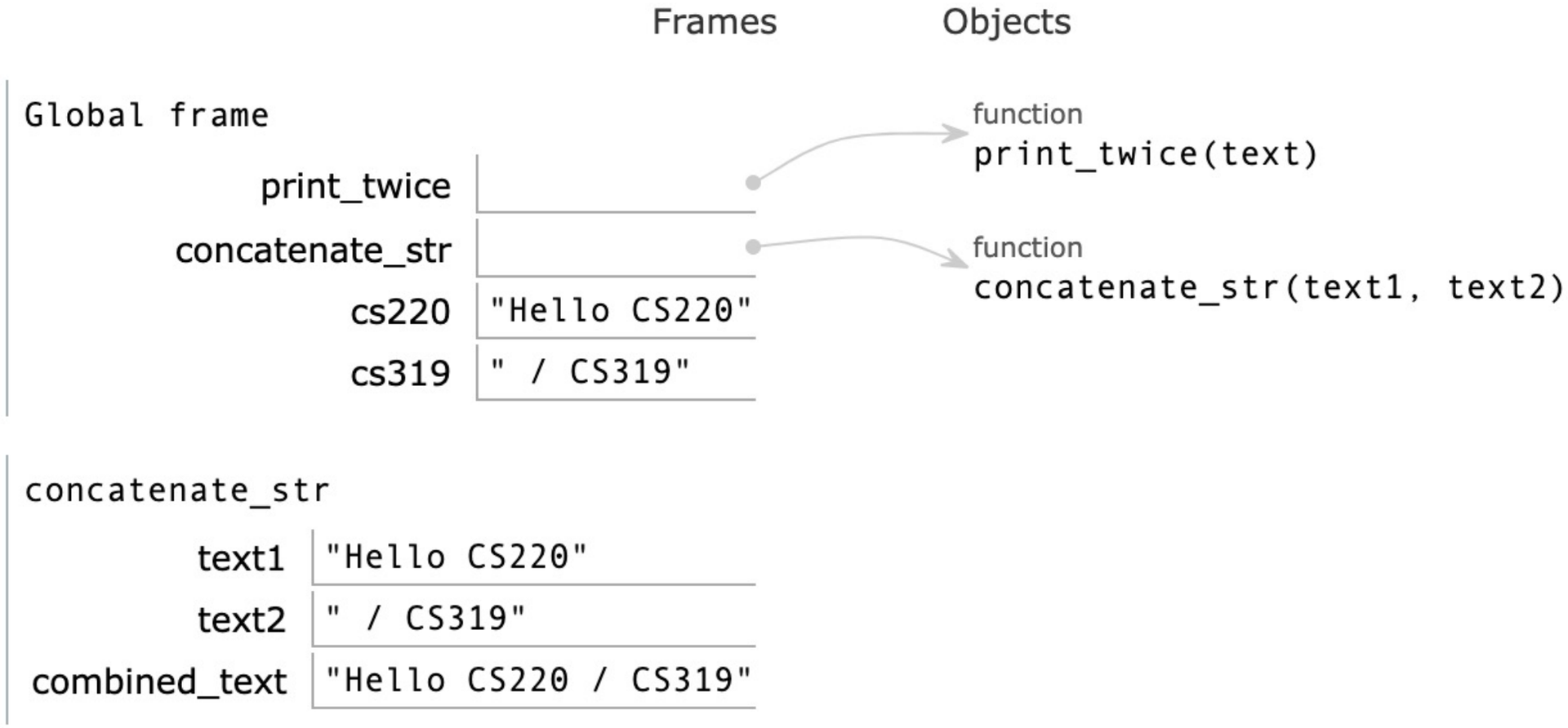
Understanding scope: example

```
1  def print_twice(text):
2      print(text)
3      print(text)
4
5  def concatenate_str(text1, text2):
6      combined_text = text1 + text2 # concatenation
7      print_twice(combined_text)
8
9  cs220 = "Hello CS220"
10 cs319 = " / CS319"
11 concatenate_str(cs220, cs319)
```



Arguments are copied to parameters:
this is called "pass by value"

Understanding scope: example (PythonTutor)



Understanding scope: example

```
def print_twice(text):
    print(text)
    print(text)

def concatenate_str(text1, text2):
    combined_text = text1 + text2
# concatenation
    print_twice(combined_text)

cs220 = "Hello CS220"
cs319 = " / CS319"
concatenate_str(cs220, cs319)
```

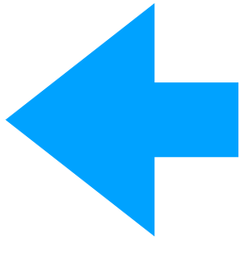
Try this example yourself using PythonTutor

Today's Outline

Context

Frames

Demos: Local Variables



Demos: Global Variables

Demos: Argument Passing

Let's do some examples in PythonTutor

Lessons about Local Variables

```
def set_x():
    x = 100

print(x)
```

Lesson 1: functions don't execute unless they're called

Lessons about Local Variables

```
def set_x():
    x = 100

set_x()
print(x)
```

Lesson 2: variables created in a function die after function returns

Lessons about Local Variables

```
def count():
    x = 1
    x += 1
    print(x)

count()
count()
count()
```

Lesson 3: variables start fresh every time a function is called again

Lessons about Local Variables

```
def display_x():
    print(x)

def main():
    x = 100
    display_x()

main()
```

Lesson 4: you can't see the variables of other function invocations, even those that call you

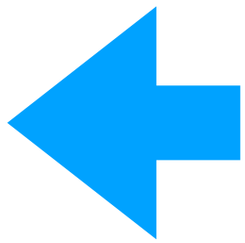
Today's Outline

Context

Frames

Demos: Local Variables

Demos: Global Variables



Demos: Argument Passing

Lessons about Global Variables

```
msg = 'hello' # global, outside any func

def greeting():
    print(msg)

print('before: ' + msg)
greeting()
print('after: ' + msg)
```

Lesson 5: you can generally just **use** global variables inside a function

Lessons about Global Variables

```
msg = 'hello'

def greeting():
    msg = 'welcome!'
    print('greeting: ' + msg)

print('before: ' + msg)
greeting()
print('after: ' + msg)
```

Lesson 6: if you do an assignment to a variable in a function, Python assumes you want it local

Lessons about Global Variables

```
msg = 'hello'

def greeting():
    print('greeting: ' + msg)
    msg = 'welcome!'

print('before: ' + msg)
greeting()
print('after: ' + msg)
```

Lesson 7: assignment to a variable should be before its use in a function, even if there's a global variable with the same name

Lessons about Global Variables

```
msg = 'hello'

def greeting():
    global msg
    print('greeting: ' + msg)
    msg = 'welcome!'

print('before: ' + msg)
greeting()
print('after: ' + msg)
```

Lesson 8: use a global declaration to prevent Python from creating a local variable when you want a global variable

Today's Outline

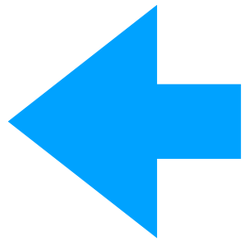
Context

Frames

Demos: Local Variables

Demos: Global Variables

Demos: Argument Passing



Lessons about Argument Passing

```
def f(x):
    x = 'B'
    print('inside: ' + x)

val = 'A'
print('before: ' + val)
f(val)
print('after: ' + val)
```

Lesson 9: in Python, arguments are "passed by value", meaning
reassignments to a parameter don't change the argument outside

Lessons about Argument Passing

```
x = 'A'

def f(x):
    x = 'B'
    print('inside: ' + x)

print('before: ' + x)
f(x)
print('after: ' + x)
```

Lesson 10: it's irrelevant whether the argument (outside) and parameter (inside) have the same variable name

Lesson Summary

Local

Lesson 1: functions don't execute unless they're called

Lesson 2: variables created in a function die after function returns

Lesson 3: variables start fresh every time a function is called again

Lesson 4: you can't see the variables of other function invocations, even those that call you

Global

Lesson 5: you can generally just **use** global variables inside a function

Lesson 6: if you do an assignment to a variable in a function, Python assumes you want it local

Lesson 7: assignment to a variable should be before its use in a function, even if there's a a global variable with the same name

Lesson 8: use a global declaration to prevent Python from creating a local variable when you want a global variable

Parameters

Lesson 9: in Python, arguments are "passed by value", meaning reassignments to a parameter don't change the argument outside

Lesson 10: it's irrelevant whether the argument (outside) and parameter (inside) have the same variable name