Discussion 1

Control and Environment Diagrams

Antonio Kam
`anto [at] berkeley [dot] edu`

Announcements

- First HW and Project gets released at the start of next week
- Technical Office Hours (cs61a.org/office-hours/)
  - For help with additional setup from Lab 00 ➕
  - 1:00 PM - 4:00 PM in Cory 521 🧪
  - 4:00 PM - 6:00 PM online 💻
- I talk very fast please scream at me if I'm talking too fast

Questions/comments from last section

- What do lab rooms have?
  - They all come with computers running Linux (I *think* Ubuntu) 💻
  - You all should have after-hours access to the lab rooms if you need to use those computers!
- How fast are you at cubing?
  - Competition Times:
    - One-Handed: 6.85 Single, 10.48 Average
    - 3x3 Blindfolded: 18.38 Single, 22.06 Average
- What sort of involvements do you have with music?
  - Composing/Arranging Music
  - Piano/Voice + other instruments, but I suck at them

Temperature Check 🌡️

- Expressions (e.g. `1 + 2`, `10 ** 3`, `7 // 2`)
- Call Expressions (call expression order, what they look like)
- Values/Types (Integers, Strings, Floats, Booleans, etc.)
- Environment Diagrams
- Functions
- Control (not covered in lecture yet)
    - `if`, `while`

Agenda 🐱

- Mini-lecture on control (Booleans, boolean operators, and `if` statements)
- Questions on control
  - `Q1`, `Q2`
- Mini-lecture on `while` loops
- Questions on `while` loops
  - `Q5`, `Q6`
- Mini-lecture on environment diagrams
- Questions on environment diagrams
  - `Q7`, `Q8`

Slides can all be found on `teaching.rouxl.es`

Control 🐈

Booleans

Falsey	Truthy
`False`	`True`
`None`	Everything else
`0`	
`[]`, `""`, `()`, `{}`	

This is Python specific! This table above doesn't necessarily apply to other languages (and later into the semester you'll see an example where the table above doesn't match the way the language works)

Boolean Operators

- `not`
  - Returns the *opposite* truthy value of the expression.
    - For example, if you type `not 0`, Python will return `True`
- `and`
  - Short circuits if it reaches a *falsey* value, and returns that value
    - *This is not necessarily `False`*
  - If all the values are truthy, the *last* value is returned
- `or`
  - Short circuits if it reaches a *truthy* value, and returns that value
    - *This is not necessarily `True`*
  - If all the values are falsey, the *last* value is returned.

Short Circuiting

- In call expressions, *everything* is evaluated from left to right, but this is not the case for when things short circuit.
- In `and` and `or` statements, all statements are not necessarily evaluated.
- `and` will keep on evaluating from left to right until it finds the first *falsey* value. If it finds a *falsey* value, it simply returns that value

Short Circuiting

Examples

- `1 and True and 1/0` This will error 😭
- `1 and True and 0 and 1/0` Returns 0 ✅
- `1 or True or 1/0`
  - Returns 1 ✅
- `0 or 1 or True or 1/0`
  - Returns 1 ✅

If Statements

```
if <condition>:
    <block of statements>
[elif <condition>:] # optional; short for 'else if'
    <block of statements>
[else:] # optional
    <block of statements>
```

- Don't forget the colons!
- `else` does not need a conditional
- You can chain together as many `elif` blocks as you want
- Evaluate all `if`s unless there's a `return` statement that ends the function
    - If you have a whole block of if/elif/else, you only evaluate at maximum 1 of the blocks.

Example (If Statements)

```
n = 0
if n == 0:
    print("hi")
else:
    print("bye")
if n == n:
    print("0")
```

In this case, the console will output

```
hi
0
```

Question 1 (2 Minutes)

What's the result of evaluating the following code?

```python
def special_case():
    x = 10
    if x > 0:
        x += 2
    elif x < 13:
        x += 3
    elif x % 2 == 1:
        x += 4
    return x

special_case()
```

- Answer: 12

Question 1 (2 Minutes)

What's the result of evaluating the following code?

```python
def just_in_case():
    x = 10
    if x > 0:
        x += 2
    if x < 13:
        x += 3
    if x % 2 == 1:
        x += 4
    return x

just_in_case()
```

- Answer: 19

Question 1 (2 Minutes)

What's the result of evaluating the following code?

```python
def case_in_point():
    x = 10
    if x > 0:
        return x + 2
    if x < 13:
        return x + 3
    if x % 2 == 1:
        return x + 4
    return x

case_in_point()
```

- Answer: 12 (Return will *prematurely* exit a function 🙀)

Question 1

Notice how the first version and the third version of our function did the same thing?

- Question: When do you think using a series of `if` statements has the same effect as using `if` and `elif` blocks?
- Answer: A series of `if` statements has the same effect as using both `if` and `elif` cases if each `if` block ends in a `return` statement of some sort (that functionally acts the same)

Question 2 (5 Minutes)

Alfonso will only wear a jacket outside if it's below 60 degrees 🥶 or if it's raining ☔.

Write a function that takes in the current temperature `temp` (integer), and a boolean signifying whether it is `raining` or not.

Try to write this function both with an if statement and in a single line! Hints on how to do this are on the worksheet.

```
def wears_jacket_with_if(temp, raining):
    """
    >>> wears_jacket_with_if(90, False)
    False
    >>> wears_jacket_with_if(40, False)
    True
    >>> wears_jacket_with_if(100, True)
    True
    """
```

Question 2

```
def wears_jacket_with_if(temp, raining):
    # Version 1
    if temp < 60 or raining:
        return True
    else:
        return False

    # Version 2
    return temp < 60 or raining
```

While Loops

```
while <condition>:
    <block of statements>
```

- A while loop allows for a repeated execution of a certain block of code, allowing you to write just one thing that will end up being executed multiple times.
- The condition is checked before the execution of each *iteration*.
- To avoid an infinite loop, you must make sure your while loop changes the variable in the condition

While Loops Examples

Example 1

```python
n = 0
while n < 5:
    print(n)
    n = n + 1 # Without this line you will have an infinite loop!
print(n)
```

Output:

```
0
1
2
3
4
5
```

While Loops Examples

Example 2

```
n = 5
while n < 5:
  # Doesn't pass the condition on the initial loop
  # as a result, doesn't run any of the blocks
    print(n)
    n = n + 1 # Without this line you will have an infinite loop!
print(n)
```

Output:

```
5
```

Attendance

`links.rouxl.es/disc`

Question 4 (5 Minutes)

Write a function that returns `True` if a positive integer `n` is a prime number and `False` otherwise.

> **Hint**: Use the `%` operator: `x % y` returns the remainder of x when divided by y.

```python
def is_prime(n):
    """
    >>> is_prime(10)
    False
    >>> is_prime(7)
    True
    """
```

Question 4

```
def is_prime(n):
    """
    >>> is_prime(10)
    False
    >>> is_prime(7)
    True
    """
    if n == 1:
      return False
    k = 2
    while k < n:
      ...
```

What should I put in the while loop? Where should my return values be?

Question 4

```python
def is_prime(n):
    """
    >>> is_prime(10)
    False
    >>> is_prime(7)
    True
    """
    if n == 1:
      return False
    k = 2
    while k < n:
      if n % k == 0:
        return False
      k = k + 1
    return True
```

Question 5 (5 minutes)

Implement the `fizzbuzz` sequence, which prints out a single statement for each number from 1 to `n`.

- If `i` is divisible by 3 (only), print `"fizz"`
- If `i` is divisible by 5 (only), print `"buzz"`
- If `i` is divisible by 15, print `"fizzbuzz"`
- Else print the number `i` itself
- No return value

Things to note:

- If a number is divisible by 15, it is also divisible by 3 and 5 - how do you take account for that?

Question 5

```
def fizzbuzz(n):
  i = 1
  while i <= n: # n inclusive
    if i % 15 == 0:
      print("fizzbuzz")
    elif i % 5 == 0:
      print("buzz")
    elif i % 3 == 0:
      print("fizz")
    else:
      print(i)
```

What's wrong with the code above?

Question 5

```python
def fizzbuzz(n):
  i = 1
  while i <= n: # n inclusive
    if i % 15 == 0:
      print("fizzbuzz")
    elif i % 5 == 0:
      print("buzz")
    elif i % 3 == 0:
      print("fizz")
    else:
      print(i)
    i += 1 # Shorthand for i = i + 1
```

Environments 🌎

Environment Diagrams

- Environment diagrams are a great way to learn how coding languages work under the hood
- Keeps track of all the variables that have been defined, and the values that they hold
  - Done with the use of *frames*
- Expressions evaluate to values:
  - `1 + 1` → `2`
- Statements do not evaluate to values:
  - `def` statements, assignments, etc.
- Statements change our environment

Frames

- The `Global Frame` exists by default
- Frames list bindings between variables and their values
- Frames also tell us how to look up values

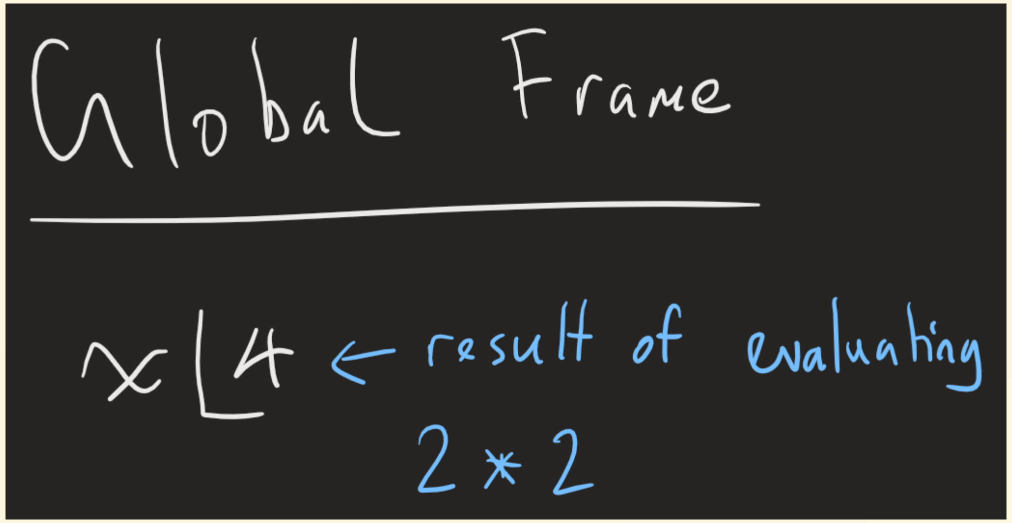
Assignment

- Assignment statements bind a value to a name
  - The right side is evaluated before being bounded to the name on the left
  - `=` is not the same in Python and mathematics
- These are then put in the *correct frame* in the environment diagram

```
x = 2 * 2 # 2 * 2 is evaluated before bound to the name x
```

Assignment

```
x = 2 * 2 # 2 * 2 is evaluated before bound to the name x
```



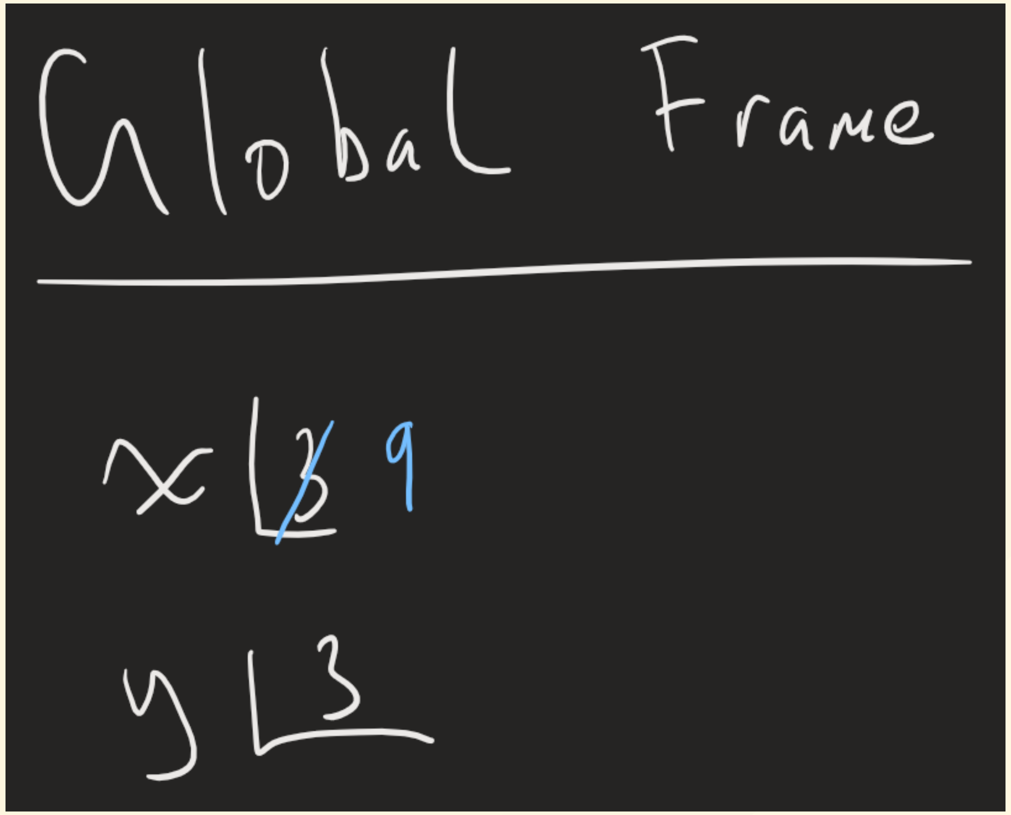
Question 7 (5 minutes)

Draw out an environment diagram for the following statements:

```
x = 11 % 4
y = x
x **= 2
```

- For future reference, you can use Python Tutor to verify your solutions!

Question 7



`def` statements

- Creates function (objects), and binds them to a variable name
- The function is **not** executed until called!
- Name of the variable is the name of the function
- Parent of the function is the frame where the function is *defined*
- Keep track of:
  - Name
  - Parameters
  - Parent

Example

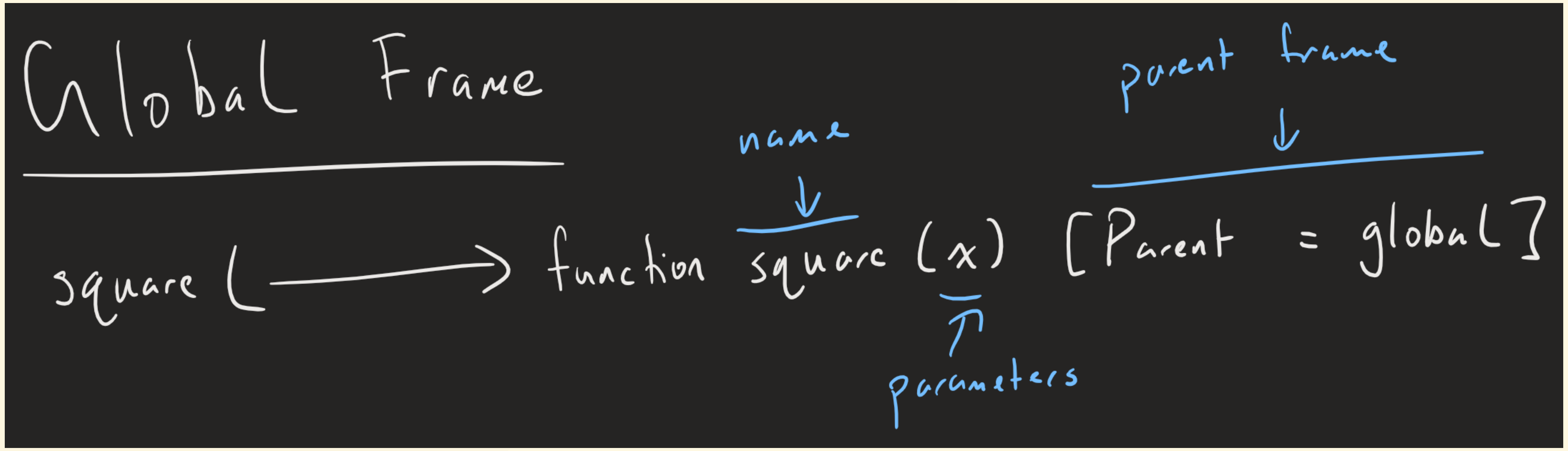
```python
def square(x):
  return x * x
```

- Keep track of the name, parameters, and parent!
- Uses *pointers* (unlike for primitive values)

Example

```python
def square(x):
  return x * x
```

- Keep track of the name, parameters, and parent!
- Uses *pointers* (unlike for primitive values)



Question 8 (5 minutes)

Draw the environment diagram for the following:

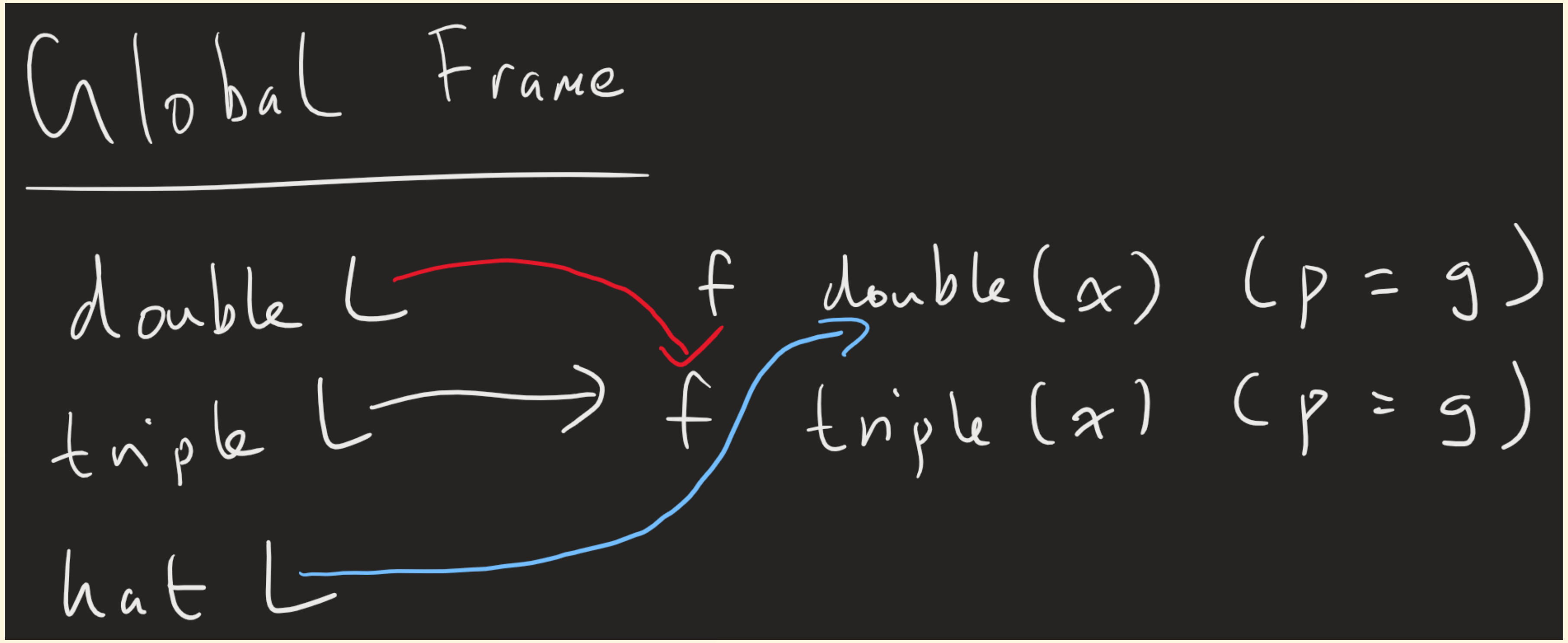
```
def double(x):
    return x * 2

def triple(x):
    return x * 3

hat = double
double = triple
```

- Start off with defining `double` and `triple`, then figure out what to do from there 👀

Question 8



Mental Health Resources

- CAPS:
    - If you need to talk to a professional, please call CAPS at 510-642-9494.
- After Hours Assistance
    - For any assistance after hours, details on what to do can be found at this link

Anonymous Feedback Form

`links.rouxl.es/feedback`

Thanks for coming! 🥳

Please give me feedback on what to improve!