Discussion 01

Control, Environment Diagrams

Aditya Balasubramanian
aditbala [at] berkeley [dot] edu

Slides available at `teaching.aditbala.com`

Announcements 📣

- Technial OH 1-4pm in Cory 521 on Thursday (6/23) and 1-3pm in Cory 521 on Friday (6/24)
- Some appointment based OH on Thursday and Friday and signups for those will occur at midnight the night before at `oh.cs61a.org` . The schedule for OH can also be found at `https://cs61a.org/office-hours/`

All Slides can be found on

`teaching.aditbala.com`

Control

Booleans

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

Some `<conditional expressions>` that will evaluate to either `False` / `True` most of the time, with a few exceptions later into the semester.

Boolean Operators

- `not <conditional expression>`
  - returns opposite of `<conditional expression>`
  - `not (1 == 2)` -> `True`
- `<conditional expression> or <conditional expression>`
  - returns the first **Truthy** value it finds, `False` if none
  - `0 or None or 1` -> `1`
- `<conditional expression> and <conditional expression>`
  - return first **Falsey** value, or last value if everything is true
  - `40 and 0 and True` -> 0
  - `40 and 1 and True` -> True

Short Circuiting

- Sort of like making an assumption
  - If I'm broke, then I don't need to check the price of boba since I'll never be able to buy it **lol** 😢
- `and` will stop at the first **Falsey** value and return it
- `or` will stop at the first **Truthy** value and return it
- Why is this important?
  - May not need to evaluate all expressions. Even if there is an expression that errors, e.g. `1/0`, `and` / `or` expression might short circuit before it reaches error

Boolean Examples

- `0 or 435 or False`
  - returns `435`
- `True and "Hello" and 0`
  - returns `0`
- Short Circuiting
- `3 and 1/0 and False`
  - returns `Error`
- `3 and False and 1/0`
  - returns `False`

If Statements

- How to use `<conditional expressions>` to execute/skip lines of code?

```
if <conditional expression>:
    <suite of statements>
elif <conditional expression>:
    <suite of statements>
else:
    <suite of statements>
```

- Colons after `if`, `elif`, `else` statements
- `else` doesn't need `<conditional expression>`

If Statements Example

```
wallet = 0

if wallet > 0:
    print('you are not broke')
else:
    print('you are broke')
if wallet == 0:
    print(0)
```

If Statements Example

```
wallet = 0

if wallet > 0:
    print('you are not broke')
else:
    print('you are broke')
if wallet == 0:
    print(0)
```

```
you are broke
0
```

General Tips for Approaching Problems

- Do not immediately start coding
  - Ensure you understand the problem
  - Have an idea of what you want to code
- Groupwork
  - Bounce ideas off of each other!
  - Share any ideas, questions, or misconceptions
- Reading the problem
  - Please read the entire problem
  - Hints are very useful
  - Doctests are SUPER useful

Question 1 (2 minutes)

```
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()
```

Question 1 (2 minutes)

```
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()
```

Question 1 (2 minutes)

```
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()
```

Question 2 (5 minutes)

```
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
    """
    "*** YOUR CODE HERE ***"
```

While Loops

- How to execute a statement multiple times in a program?

```
while <conditional clause>:
    <statements body>
```

- program executes until `<conditional clause>` is false
- In other words, only run when `<conditional clause>` evaluvates to `true`

While Loop Examples

```
x = 3
while x > 0:
    print(x)
    x -= 1
```

While Loop Example

```
x = 3
while x > 0:
    print(x)
    x -= 1
```

```
3
2
1
```

While Loop Example

- What is wrong with this while loop

```
x = 3
while x > 0:
    print(x)
```

- This will result in an infinite loop
- Make sure you are modifying the condition in the while loop

Question 4: Is Prime? (10 min)

Hint: Use the % operator: x % y returns the remainder of x when divided by y

```
def is_prime(n):
    """
    >>> is_prime(10)
    False
    >>> is_prime(7)
    True
    >>> is_prime(1) # one is not a prime number!!
    False
    """
    "*** YOUR CODE HERE ***"
```

Question 5: Fizzbuzz (15 min)

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

- If `i` is divisible by 3 only, then we print `fizz`.
- If `i` is divisible by 5 only, then we print `buzz`.
- If `i` is divisible by both 3 and 5, then we print `fizzbuzz`.
- Otherwise, we print the number `i` by itself.

Enviroment Diagrams 

Enviroment Diagrams

- What are they?
  - A way to model how our program runs line by line
  - Keep track of variables, function calls and what they return, etc.
- Why use them?
  - Can help us understand where there is a bug in program (debugging)
  - Useful for other questions (WWPD, coding)
  - Exam points!

Important Concepts

- Expressions
  - Evaluate to values
  - `1 + 1` -> `2`
- Statements
  - Bind **names** to **values**
  - **Names**
    - `def` statements, assignment statements, variable names
  - **Values**
    - numbers, strings, functions, or other objects
  - `x = 2`
  - doesn't return anything

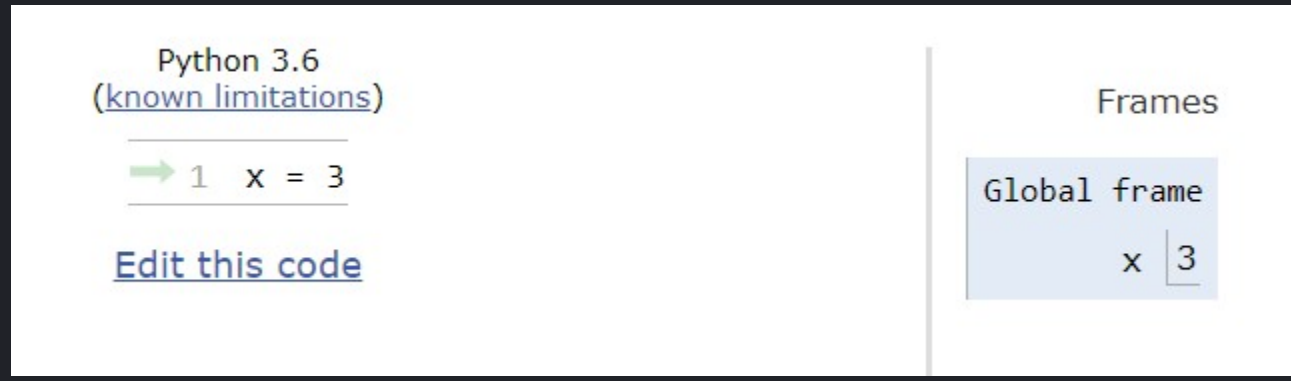
Interactive Example

```
x = 3

def square(x):
    return x ** 2

square(2)
```

- Let's create an enviroment diagram for this program!
- Start from top, go to bottom

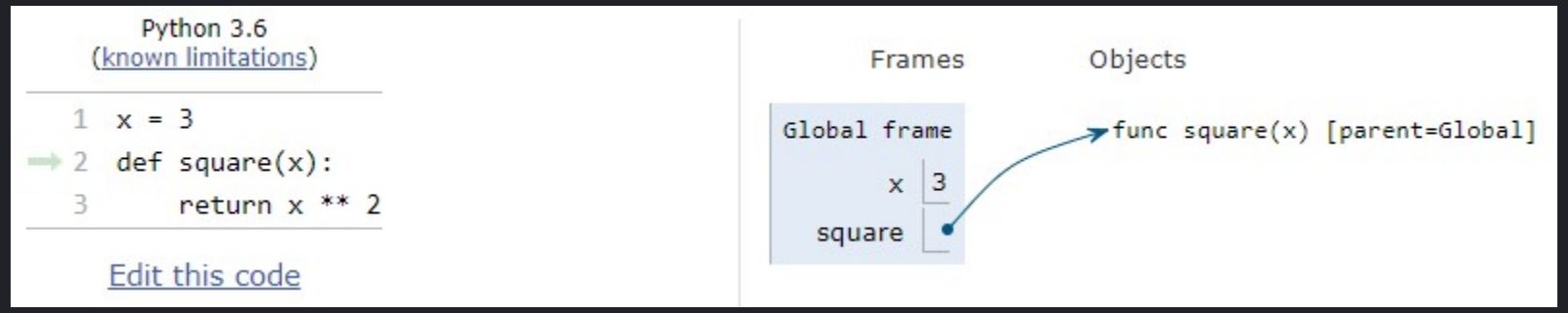
Frame

- Frames are objects that list bindings of **variables** and **values**
  - tell us how to look up bindings
- **Global Frame** exists by default
- Assignment statement (denoted by =) creates binding of **variable** `x` and **values** `3`



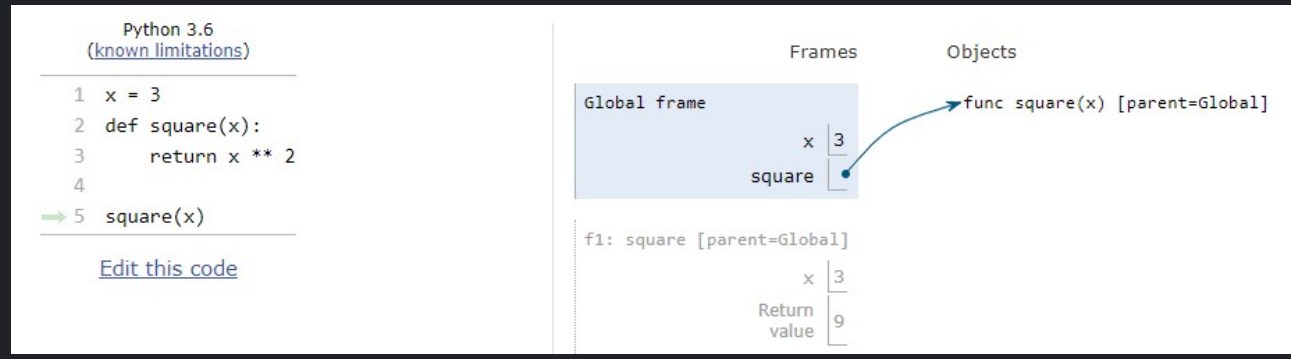
`def` statements

- `def` statements are used to bind **function objects** to a **variable**
- Only bind, **NO** execution until function is called
  - `def foo():` -> define function called `foo` with no parameters
  - `foo()` -> execute foo
- Binding name is function name
- Parent function is frame where function is defined
- Keep track of *name*, *parameters*, *parent frame*



Call Expressions

- Syntax: `function_name(arg1, arg2, ...)`
- Create new frame for call expression
- Steps for evaluating:
  1. Evaluate operator (function)
     - See if it exists
  2. Evaluate operands (args)
     - simplify args
  3. Apply operator to the operands



[Study Groups???]

Thank you!