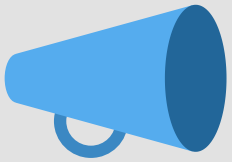
Discussion 01

Control, Environment Diagrams

Aditya Balasubramanian
aditbala [at] berkeley [dot] edu

Slides available at `teaching.aditbala.com`

Announcements 

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` -> False
  - `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
```

[ADD EXAMPLES]

While Loops

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

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

- program executes until `<conditional clause>` is false

[ADD EXAMPLES]

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)
```

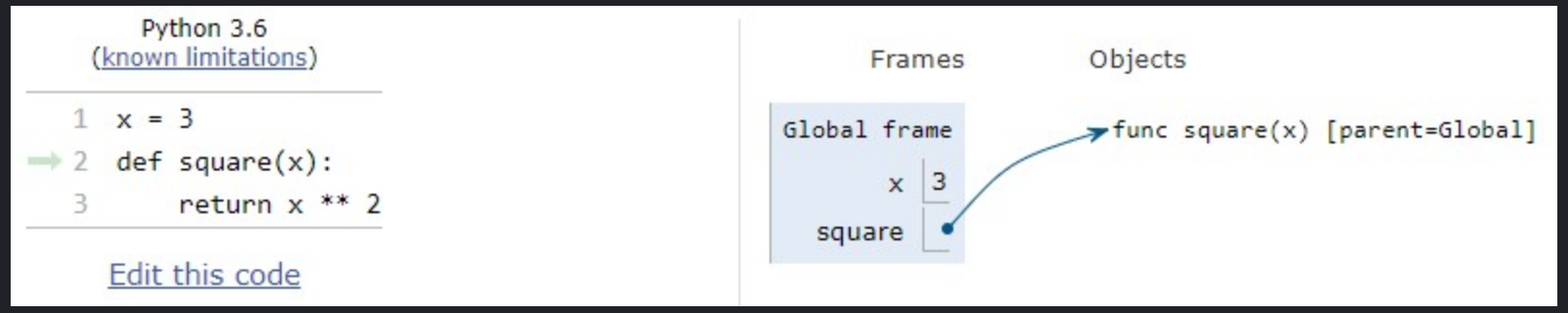
- Let's create an enviroment diagram for this program!
- Start from top, go to bottom

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`

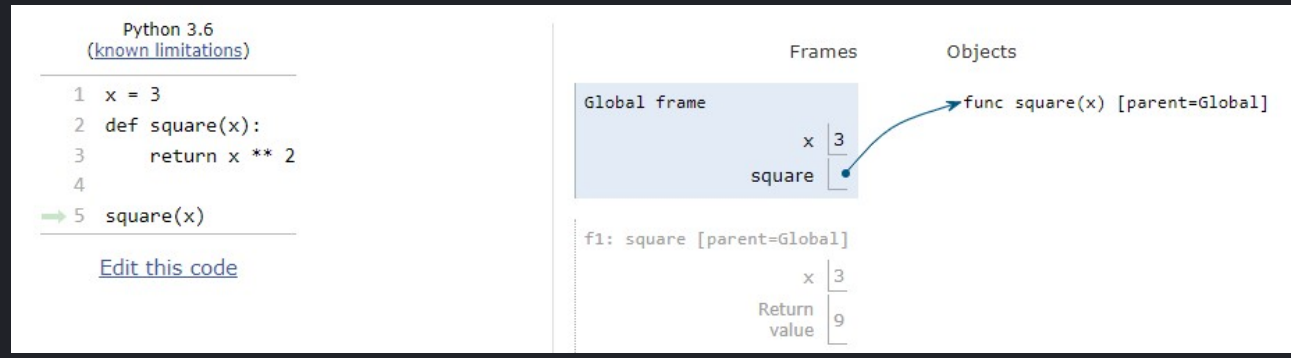
`def` statements

- `def` statements are used to bind **function objects** to a **variable**
- Only bind, **NO** execution until function is called
  - `def foo():`
  - `foo()` -> execution
- 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



Thank you!