Discussion 01: Control, Environments, and HOFs

TA: **Jerry Chen**
Email: **jerry.c@berkeley.edu**
TA Website: **jerryjrchen.com/cs61a**

Agenda

1. Attendance
2. Announcements
3. Booleans & Control (skipped, view slides later)
4. Environments
5. Higher Order Functions

Attendance

Sign in at bit.do/jerrydisc

OR

Please put your name, SID, and email on the sign-in sheet.

Announcements

Make sure you're registered on OK

Hog released due next Thursday

HW 1 due Thursday (Today!!!)

HW 2 released (vitamin only) due Tuesday

Lab 1 due Friday

Project Partner Mixer is Thursday (today) 12:30pm in 430 Soda

- Project partner finding thread on Piazza is open!

Booleans

- There are “truthy” and “falsy” values:

“Truthy”	“Falsy”	Notes
`True`	`False`	
`“banana”`	`‘’`	Empty string
`100, -12`	`0`	
`[1, 2, 3], {‘a’: 1, ‘b’: 2}`	`[], {}`	Will see later in the course

Boolean Operators

- **not** (negates)**,**
- **and** (true iff both are true)**,**
- **or** (false iff both are false)

- **Short circuit** and terminate early once the **result of a expression is known**

Control

If statements

```
if <exp>:
     <suite>
elif <exp>:
     <suite>
...
elif <exp>:
     <suite>
else:
     <suite>
```

Careful!

```
if <exp>:
     <suite>
if <exp>:
     <suite>
else:
     <suite>
```

Control

While statements

- The **expression is checked before** executing the suite

```
while <exp>:
    <suite>
```

Control

A programmer's spouse tells them, "While you're at the grocery store, buy some eggs." They never come back.

A programmer's spouse asks them, "Please go the to store and buy a loaf of bread. If they have eggs, buy a dozen."

They come back with a dozen loaves of bread.

FizzBuzz

Write a program that prints the numbers from 1 to n. But:

- For **multiples of three print “Fizz”** instead of the number.
- For the **multiples of five print “Buzz”**.
- For **numbers which are multiples of both three and five print “FizzBuzz”**.

FizzBuzz

Solution might look something like this:

```
def fizzbuzz(n):
    i = 1
    while i <= n:
        if i % 3 == 0 and i % 5 == 0:
            print("FizzBuzz")
        elif i % 3 == 0:
            print("Fizz")
        elif i % 5 == 0:
            print("Buzz")
        else:
            print(i)
        i += 1
```

FizzBuzz

EnterpriseQualityCoding / **FizzBuzzEnterpriseEdition**

Watch 132 | Star 4,846 | Fork 269

Code | Issues 119 | Pull requests 22 | Wiki | Pulse | Graphs

Tree: **00097f...** | Find file | Copy path

FizzBuzzEnterpriseEdition / src / main / java / com / seriouscompany / business / java / fizzbuzz / packagenamingpackage / impl / parameters / **DefaultFizzBuzzUpperLimitParameter.java**

emiln Merge branch 'feature/Dependency-injection' `00097ff` on Apr 19, 2015

2 contributors

17 lines (10 sloc) | 519 Bytes | Raw | Blame | History

```java
package com.seriouscompany.business.java.fizzbuzz.packagenamingpackage.impl.parameters;

import org.springframework.stereotype.Service;

import com.seriouscompany.business.java.fizzbuzz.packagenamingpackage.interfaces.parameters.FizzBuzzUpperLimitParameter;

@Service
public class DefaultFizzBuzzUpperLimitParameter implements FizzBuzzUpperLimitParameter {

        public int obtainUpperLimitValue() {
                return DefaultFizzBuzzUpperLimitParameterValue;
        }

        private final int DefaultFizzBuzzUpperLimitParameterValue = 100;
}

```

Environments

Q: What is an **environment?**

A: Environments represent a **context** for execution.

- Environments store things such as name-value bindings

- Visualize environments using **environment diagrams**

Environment Diagrams

Consists of many frames that track program state

Some rules:

- **Function call: create and number new frame** (f1, f2, etc.) — always start in global frame

- **Assignment:** write variable name and expression value

- **Def statements:** record function name and bind function object. Remember parent frame!

- **Frames return values** upon completion (Global is special)

Higher Order Functions

Big idea: **Functions can be treated as "variables" — a powerful tool for abstraction!**

- Can pass as arguments or returned
- Analogy is a bit limited, can't necessarily "add" two functions

Functions that manipulate other functions are **higher order**

Higher Order Functions

Packager Example

```
def make_packager():
    def packager(item):
        return "[[[" + item + "]]]"
    return packager

p = make_packager()
print(p("toothbrush"))
```

Higher Order Functions

Id Example

```
def id(x):
    return x

print(id(id)(id(13)))
```