CSCI 1300

Intro to Computing

Gabe Johnson

Lecture 27 Mar 18, 2013

Review for Exam 3

Upcoming Exam

Wed, March 20

Exam 3

This exam covers Python and Java. We'll give examples in Python, and ask you to use what you've learned about Java to solve similar problems.

Test Format

50 Minutes
Take a test from the top of the stairs
Get started

Open Notes
Closed Gizmos

Compare Python vs. Java

```
def burninate(data, val):
    for i in range(len(data)):
        data[i] = data[i] * val

orig = [ 6, 9, 0, 2 ]
print "Before burninating:"
for item in orig:
    print item
burninate(orig, 2)
print "After burninating:"
for item in orig:
    print item
```

Before burninating:

6

9

0

2

After burninating:

12

18

0

4

Same Program in Java

```
public class Burninate {
    public static void main(String[] args) {
        int[] orig = new int[] { 6, 9, 0, 2 };
        System.out.println("Before burninating:");
        for (int i : orig) {
        System.out.println(i);
        }
        burninate(orig, 2);
        System.out.println("After burninating:");
        for (int i : orig) {
        System.out.println(i);
        }
    }

    static void burninate(int[] data, int val) {
        for (int i=0; i < data.length; i++) {
        data[i] = data[i] * val;
        }
    }
}
```

Before burninating:
6
9
0
2
After burninating:
12
18
0
4

Python code = hints

We will ask you to read and comprehend some Python code and give you the output. This way you can see for certain how the code works.

Then we'll give you a similar (but not exact) Java program and ask you to tell use what the output is.

This is about code comprehension.

Python code is interpreted

Python code is interpreted. This means the interpreter reads your source code line by line, and records what it sees as it moves along.

If you try to read a variable or call a function before the interpreter knows what it is, you'll get a Name Error.

While Loops

In *both* languages, a while loop will execute the entire block, unless you explicitly tell it to break or or return. So in these loops, the final value of x is 4, not 3.

```
int x = 0;
while (x < 4) {
    x = x + 1;
}
System.out.println(x);
```

```
x = 0
while x < 4:
    x = x + 1
print(x)
```

Go Beyond Python

```
count = 0
sum_of_odds = 0
sum_of_evens = 0
num_odds = 0

data = [5, 7 ,8, 9, 10, 16, 15, 0, 1]
for d in data:
    count = count + 1
    if d % 2 == 1:
        sum_of_odds = sum_of_odds + d
        num_odds = num_odds + 1
    else:
        sum_of_evens = sum_of_evens + d

print "Count", count
print "Sum of Odds", sum_of_odds
print "Sum of Evens", sum_of_evens
print "Num odds:", num_odds
```

Here's Python code. It prints out:

Count 9
Sum of Odds 37
Sum of Evens 34
Num odds: 5

Here is *similar* Java code

```
public class WhileThing {
  public static void main(String[] args) {
    int count = 6;
    int sum_of_odds = -10;
    int sum_of_evens = 0;
    int prod_evens = 0;

    int[] data = new int[] {5, 7 ,8, 9, 10, 16, 15, 0, 1 };

    for (int d : data) {
        count = count + 1;
        if (d % 2 == 1) {
            sum_of_odds = sum_of_odds + d;
        } else {
            sum_of_evens = sum_of_evens + d;
            prod_evens = prod_evens * d;
        }
    }
    System.out.println("Count: " + count);
    System.out.println("Sum of Odds: " + sum_of_odds);
    System.out.println("Sum of Evens: " + sum_of_evens);
    System.out.println("Prod Evens: " + prod_evens);
  }
}
```

What does *it* print?

Requirements of Java files

Every Java source file MUST:

* contain a class
* the name of the public class must be the same as the filename

Further, when compiled, `Foo.java` is turned in to `Foo.class` (and possibly other things, if there are non-public classes in `Foo.java` as well).

Java is *statically typed*

This means that whenever we have a variable or function we *must* know what type it is. While you can get away with this in Python:

x = 4
x = “Hi”

You can’t in Java, because we must say that x is either an integer or a String (or something else), and *it must remain that data type forever*.

Java Data Types

Several common data types in Java:

int: an integer (counting number): 0, 1, 2, 3...
float: number with decimals: 0.0, 0.3, 3.24...
double: same as float but higher precision
char: single character, e.g. ‘x’. Single quotes.
String: a bunch of characters “xyz”. Double quotes.
boolean: true/false value
void: indicates ‘no value’.

Java Declarations

In Java, you *declare* a variable by giving the type and name of that variable. Examples:

```
char[] prev;
char[] other = new char { ‘x’, ‘y’ };
```

Once you’ve declared a variable, you *don’t need to redeclare it*. You can simply use it:

```
prev = new char[7];
prev[3] = ‘*’;
int otherLenSquared = other.length * other.length;
```

Shadowing a variable

I saw some people doing this. When you redeclare a variable that already exists, Java says “OK! Let’s make a private version of that and ignore the other one! This will be fun!”

So if you have a variable already declared, and you hope to use it in your current context, *don’t redeclare it*, because you won’t be using the same variable. It might have the same name, but it’s not the same variable.

Java Arrays

An array in Java has a type and length. You can set the length based on some runtime information, or to a constant value known at runtime.

E.g.

```
int[] numbers = new int[some_number];
```

or

```
int[] other = new int[4];
```

or

```
int[] dyna = new int[] { 5, 4, 3, 2, 1, 0 };
```

Array Read

You can *read* an array `A` using an index that is in the range `0..A.length-1`:

```
// assume A.length > 0
int x = A[0]; // ok
int y = A[A.length-1]; // ok
int z = A[A.length]; // not ok
```

Array Write

Similarly you can write into an array in the same ranges:

```
// assume A.length > 0
A[0] = 4; // ok
A[A.length-1] = 3; // ok
A[A.length] = 2; // not ok
```