For Better & For Worse: Java

Programming Languages

Different programming languages have different advantages and drawbacks.

- Features & design decisions about a language give it different **characteristics**
- Characteristics combine to form qualitative assessments of the language's performance in important criteria:
  - **readability,**
  - **writability,**
  - and **reliability.**

Readability

Why is Readability Important?

Most code will be used *more than once* and *as part of a larger software system*.

The ability of a team to **maintain** (fix, debug, improve) a codebase is directly related to the ease with which that code can be understood.

And, for you: you have to read code examples to be able to learn how to write.

Readability & Writability

As novice programmers, the level at which you're able to read code directly feeds into the level at which you're able to write it.

One distinguishing feature of programming experience: *the ability to read (albeit at some reduced level) code that one would not be able to write oneself*.

So, Today:

Pretty much all about readability, which is directly related to writability.

Both criteria combine in some way to produce *learnability*, or *suitability for intro programming*.

What Affects Readability?

- Overall Simplicity
- Orthogonality
- Availability of Data Types
- Choice of Syntax

Simplicity and Readability

A language is simple when it contains

- few basic constructs
- low **feature multiplicity,** i.e. "one way to do each thing"
- little or controlled operator overloading

Constructs of a Language

With Java, you need:

- variables
- data types
- conditionals
- while & for loops
- functions
- arrays
- classes & objects

...and then you have the whole language, really.

Constructs of a Language

With Python, you need:

- variables
- data types
- conditionals
- while & for loops
- functions
- lists, **and then list comprehensions**
- sets, dictionaries
- classes & objects (**including functions themselves**)
- modules

Simplicity and Feature Multiplicity

For a given operation, how many ways are there that you can perform it?

Feature Multiplicity: Incrementation

Feature	Java	Python
`count = count + 1`	✅	✅
`count += 1`	✅	✅
`count++`	✅	❌
`++count`	✅	❌

Java has *so many ways* to increment a variable, whereas Python just has "the natural way" and then "the shorthand."

Feature Multiplicity: Iteration

In Java, there are two `for` loops:

```
for (int i = 0; i < a.length; i++) {
    String curr = a[i];
    doSomething(curr);
}
```

```
for (String curr : a) {
    doSomething(curr);
}
```

Feature Multiplicity: Iteration

In Python, there's only one:

```
for curr in a:
    do_something(curr)
```

(no natural way to get the index here, but if you want that, then there's a *separate* tool called `enumerate()`)

Simplicity and Operator Overloading

A syntactical element is **overloaded** when the same element can have different meanings depending on the context in which it is used.

Java has **function/method overloading** but only very limited **operator overloading**.

Python has no function overloading but extensive operator overloading.

Function Overloading and Java

In Java, it's possible to write two functions that have the same name as long as the input parameters are different:

```
public static double sum(double a, double b) { ... }
public static double sum(double[] as) { ... }
```

This is convenient & not very confusing when applied conservatively.

Function Overloading and Java

It's possible to do function overloading in Java in a way that would be very confusing and should be avoided (c.f. here)

```
class Con {
  public Con(int i, String s) {
    // Initialization Sequence #1
  }
  public Con(String s, int i) {
    // Initialization Sequence #2
  }
  public Con(Integer i, String s) {
    // Initialization Sequence #3
  }
}
```

Function Overloading and Python

Function overloading is forbidden in Python by default. Nevertheless, the allure of multiple uses for the same function can be strong...

```
def my_overloaded_function(x, initial=None, in_place=False):
    if isinstance(x, list):
        ...
    elif isinstance(x, tuple):
        ...

    if not initial:
        initial = x[0]
```

Operator Overloading and Java

Java has very limited operator overloading.

Can you think of the most significantly overloaded operator that we've discussed in Java?

Operator Overloading and Java

Java has very limited operator overloading, with `+` as a notable exception.

`+` can be used to:

- do integer addition
- do floating point (double) addition
- do String-to-String concatenation
- do to-String coercion followed by concatenation

Operator Overloading and Python

Python overloads `+` in similar ways.

`+` can be used, by default, to:

- do integer addition & floating pointaddition
- do string-to-string concatenation
- do list-to-list & general seq-to-seq concatenation
- do set intersection

Operator Overloading and Python

Despite expanded set of types where `+` applies, none of the "default" uses permit mixing types.

This would be permitted in Java but would crash the program in Python.

```
"cis" + 1100
```

Operator Overloading and Python

To define the behavior of `+` when your custom type is on the left-hand side of the operator, define an `__add__` method in your class.

```
def __add__(self, other):
    real = self.real + other.real
    imag = self.imag + other.imag
    return Complex(real, imag)

x = Complex(3, 4) # represents 3 + 4i
y = Complex(5, 6) # represents 5 + 6i
z = x + y # returns a new Complex representing 8 + 10i
```

Simplicity: A Push and Pull

Nice to have some choices for how you accomplish certain tasks. Not very nice to have too many choices.

Java, despite its burdensome syntax, accomplishes fairly well the goals of:

- having a single clear reasonable way of doing most things
- having most features do just one or a few things.

Python is similar, but while the syntax is more streamlined, the features are more numerous and are frequently ambiguous.

Orthogonality

Complicated, but represents the degree to which all of the language features can be applied to and combined with all others.

- High orthogonality means that there are relatively few exceptions to the rules that the language's features follow
- Low orthogonality means that statements that "look" similar to others may have drastically different meanings depending on the context or be banned outright.

Orthogonality and Java

Both languages are fairly good with respect to orthogonality. One notable Java shortcoming (in my opinion):

In Java:

```
System.out.println(4);
int[] arr = {4, 5, 6, 7};
System.out.println(arr); // not forbidden, but definitely not the same
```

Orthogonality and Python

In Python, when defining an operator's behavior, it's possible to define different behaviors for the left-hand and right-hand operands.

```
x = MyThing()
y = "heya!"

x + y
y + x # can return two very different results!
```

Orthogonality and OCaml

If you take CIS 1200, you'll learn about *functional programming languages*.

- In these languages, there are primitives and functions.
- Functions are the single major construct, so it's easy to be consistent (read: orthogonal) with how functions work across all kinds of inputs.

Readability & Types

A language like C doesn't have boolean types. That's bad.

```
timeout = 1; // true? false? who knows?
```

Python and Java have boolean types. That's good.

```
timeout = true; // great. we got it.
```

Readability & Syntax Design

The **syntax** is the *form* (or really the spelling) of a language.

Most of the code that you write comes from you, but the presentation of certain key constructs comes from the language itself.

Special Words & Symbols

Java has some nice special words (`for`, `class`, `if`, `while`) and some not-so-nice special symbols:

- `{` and `}` to indicate code blocks
- mandatory parenthesis around boolean expressions in control flow statements

Special Words & Symbols

Python also has nice special words (`for`, `while`, `class`, `def`, `in`) and then some symbols that are... something else:

- tabs to indicate code blocks
- mandatory `:` to indicate the block of statements that follow control flow structures

Where Both Languages Succeed

Nice choices of reserved keywords that also usually mean the same thing in every context in which they're used:

- not so much `in` for Python
- not so much `for` in Java

Where Both Languages Have Shortcomings

Whether you denote blocks of statements with curly braces or with whitespace, there's still no direct indicator *on the item itself* what block is being closed when it gets closed:

```
...
...
...
...
...
...
        else {
            x = 4;
        }
    }
...
}
```

Other Clearer Implementations

```
<td>
    {% if exam.answers %}
    <a target="_blank" href="{{ exam.answers }}">Answers</a>
    {% else %}
    Coming soon!
    {% endif %} <!-- We know what this is closing! --->
</td>
```

Darker Days of Python's Past

In Java, you definitely can't do this:

```java
int true = 4;
boolean result = true > 10; // ???????
```

But in Python, you once were able to...

```python
True = False
if True == False:
    print("AHHHH") # This would print...
```

Meaning Follows Syntax (Hopefully)

In both Java and Python, the meaning of basic language features is usually clear from their presentations.

Even the confounding `static` in Java does have one specific meaning.

In `C`, it does not...

- Either a variable that is declared at compile time, or...
- a variable that is inaccessible outside of the file in which it's written
    - i.e. `private` in Java

Speaking of Private

No such thing as variable/function privacy in Python, at least in an enforceable way.

Identifiers can be semi-hidden or mangled by preceding them with single or double underscores.