CS 1: Intro to CS

Intro to Java
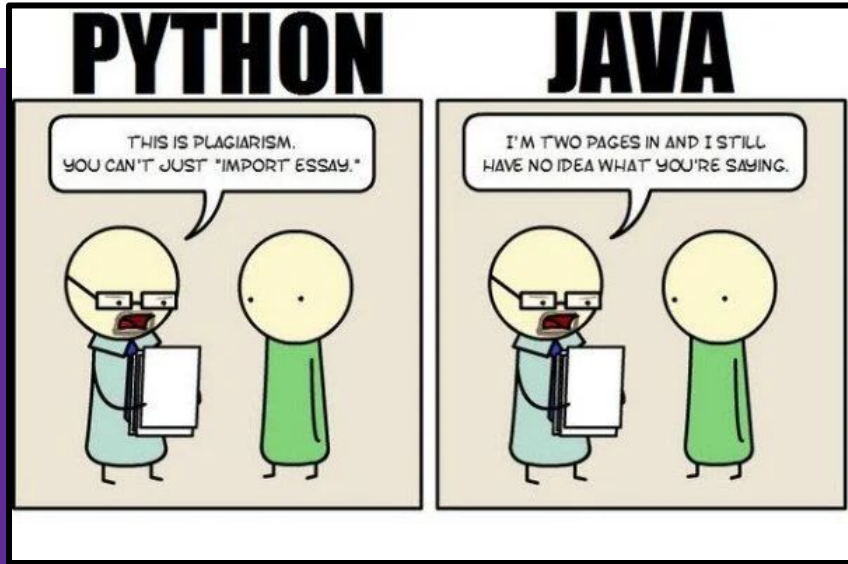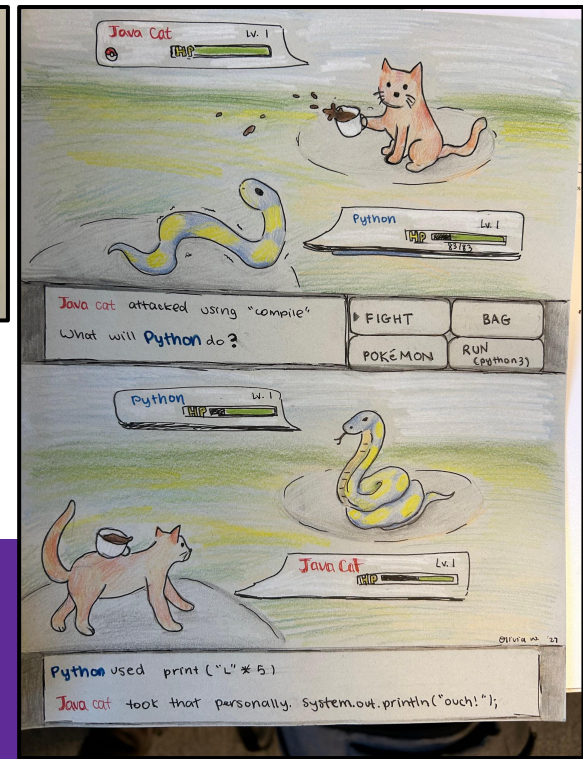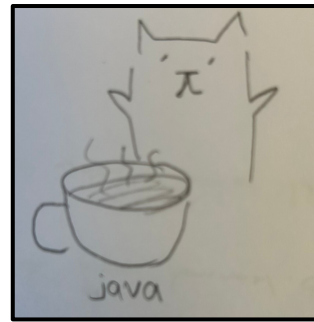
Week 8 (Combined Slide Deck)

Administrivia

MP 7 is out ("porting Python to Java" with a DNA/mRNA application!)

Intro to Java (Lectures 21-23, Lab 07)

Python vs. Java (Monday)

Compiling vs. Running (Monday)

Types (Introduced Monday)

Syntax (Introduced Monday)

Conditionals and Loops (Wednesday, Friday)

For fun: [What if Programming Languages were Pokemon?](#)

Methods (Tuesday, Wednesday)

Java Programs vs. Java Classes (Monday-Wednesday)

Tips for Week 8

In MP7A, you will first get practice with basic Java

In MP7 Parts B and C, you will practice porting a provided DNA.py class into DNA.java, and also finishing an mRNA.java and DNAClient.java to finish a fully-function DNA -> mRNA -> codon -> protein transcription feature!

To study, we encourage you to review the posted lecture slides/code (the code has comments to review!), utilizing OH, and starting MP7 early so you can come to OH with any questions

What is Java?

Java is the second programming language students see in the Caltech CS curriculum

Everything we've done in Python so far can be done in Java, but the syntax and rules are quite a bit different
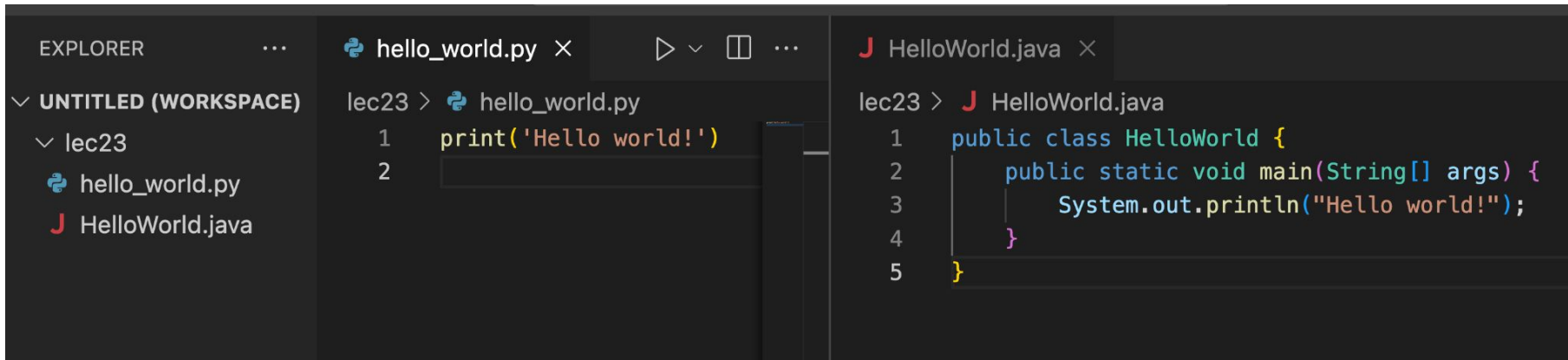
Java is a typed, compiled language (kind of like having a language translator/censor with you 24/7 yelling at you when you speak incorrectly, but once you pass the censor, you're pretty good to go out speaking in public)

Python is an interpreted language (no language translator before you speak, but after you speak incorrectly, Pythonists may yell at you, definitely judge you. Especially if use use Python 2)

What is Java, Really? A Comparison.

	Python	Java
Typing	Dynamically typed (type-checking at runtime)	Statically typed, compiled (with type-checking) then ran
Language Type	Interpreted	Compiled and interpreted
Syntax	Less syntax, "more syntactic sugar"	More verbose (and stricter) syntax
Statements	`var_name = <exp>`	`VarType varName = <exp>;`
Performance	**Compiled at run-time (slower)**	**Compiled, then executable until re-compiled (faster)**

Example: Hello World!

Example: Hello World!

These two programs are written in Python, Java respectively. What similarities and differences do you notice?

Java: Compiled and Interpreted

The most important thing to understand when starting Java is that unlike Python, we need to *compile* a Java program.

The compiling step will do syntax and type-checking and if everything checks out, a .class file will be created (or updated) with the compiled byte code

Then we can run!
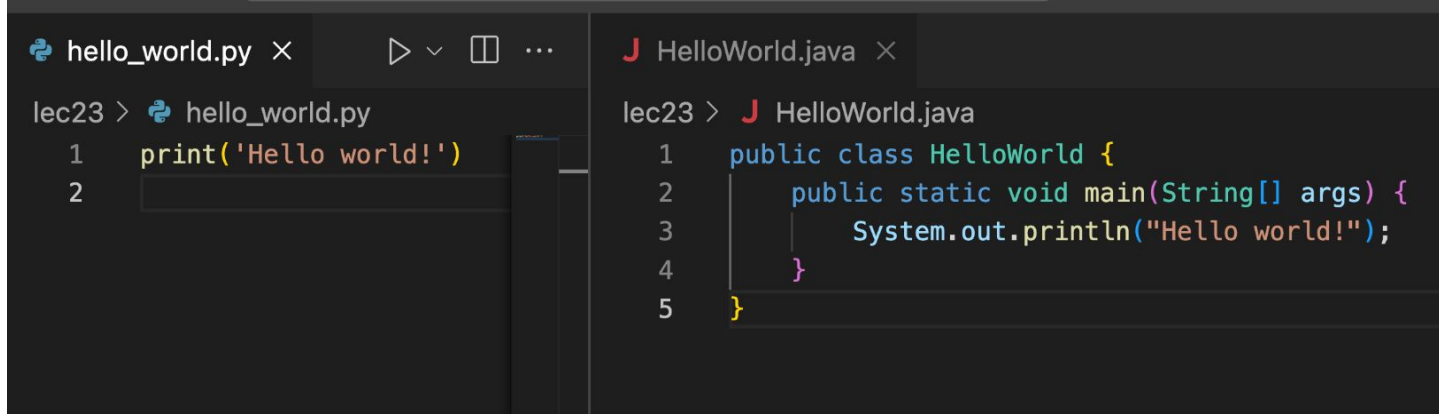
```
mehovik@Els-MacBook-Pro:~/cs1/lectures/lec23$ ls
HelloWorld.java hello_world.py
mehovik@Els-MacBook-Pro:~/cs1/lectures/lec23$ python3 hello_world.py
Hello world!
mehovik@Els-MacBook-Pro:~/cs1/lectures/lec23$ javac HelloWorld.java
mehovik@Els-MacBook-Pro:~/cs1/lectures/lec23$ ls
HelloWorld.class        HelloWorld.java        hello_world.py
mehovik@Els-MacBook-Pro:~/cs1/lectures/lec23$ java HelloWorld
Hello world!
mehovik@Els-MacBook-Pro:~/cs1/lectures/lec23$
```

But REPLs are Fun...

Recall from Reading 2 that a REPL (Read-Eval-Print-Loop) is a very convention interpreter (e.g. the Python shell) that evaluates statements at run-time.

This isn't used in practice for Java, where performance/correctness is achieved through the compilation step, but there are online REPLs (and jshell) which can be useful to quickly practice Java before getting the setup figured out. You can play around with this popular one called [replit](). **Do not rely on this for MPs, and turn off any AI auto-complete features if you do so (<u>AI tools are not allowed in CS 1</u>).**

Variables and Assignment

Variables are still assigned as *<var name> = <expression>*

Python	Java

```>>> salary = 18.5```
 	
```>>> weekly_salary = salary * 20```
 	
```>>> print(weekly_salary)```
 	
```370```
	```double salary = 18.5;```
	
	```double weeklySalary = salary * 20;```
	
	```System.out.println(weeklySalary);```
	
	```// 370```

Python	*Java*

But Java requires variables to be declared with a valid type and all statements **must** end with a **;** (semicolon)

Types (Back to Lecture 1)

Data in programming languages is subdivided into different "types":

- integers:
  - Python (**int** type): `x = 0, x = -43, x = 1001`
  - Java (**int** type): `int x = 0; int x = -43; int x = 1001;`
- floating-point numbers:
  - Python (8-byte **float** type): `x = 3.1415, x = 2.718`
  - Java (8-byte **double** type and 4-byte **float** type))
    - `double x = 3.1415; double x = 2.718;` (used in Lab 7)
    - `float x = 3.14515f; float = 2.718f;`
- boolean values:
  - Python (**bool** type): `x = True x = False`
  - Java (**boolean** type): `boolean x = true; boolean x = false;`

More Types: `Strings` vs. `char`

Strings and characters are separate types in Java (as opposed to Python having only `str`)

- Strings
    - Python (`str` type): `s ='foobar', s = 'hello, world!'`
    - Java (`String` type): `String s ="foobar"; String s = "hello, world!";`
        - *Must* be defined with `"`
- Characters:
    - Python (single-character `str`): `s ='f', s = '!'`
    - Java (`char` type): `char ch = 'f'; char ch = '!';`
        - *Must* be defined with `'`

Any many others! We won't go beyond these types (e.g. lists, dictionaries, etc.) in Java though... You can find a summary of other Java types here.

Lecture 1: Types

In Python, the same variable can hold data of different types at different times:

```
>>> a = 'foobar'
>>> a
'foobar'
>>> a = 3.1415926
>>> a
3.1415926
```

What might be an issue with this?

Java: Variables are Declared with Types

In Java, the same variable *cannot* hold data of different types after declaration.

```
>>> a = 'foobar'
>>> a
'foobar'
>>> a = 3.1415926
>>> a
3.1415926
```

```
String a = "foobar";
// "foobar"
a = 3.1415926;
// compiler error
Lec22MoreJava.java:lineNum: error:
incompatible types: double cannot be
converted to String
        a = 3.1415926;
            ^
```

Java: Variables are Declared with Types

Also, we cannot re-declare a type for an existing variable:

```
>>> a = 'foobar'          String a = "foobar";
>>> a                     // "foobar"
'foobar'                  String a = "foo";
>>> a = 'foo'             // compiler error
>>> a                     Lec22MoreJava.java:lineNum: error: variable
'foo'                     a is already defined in method
                          main(String[])
                                  String a = "foo";
                                          ^
```

Printing: `System.out.println(<exp>);`

Since we don't have a Java interpreter to dynamically evaluate and output values in the interpreter (well, there is `jshell`), we will be using `System.out.prinltn(<exp>);` to output values.

```java
1   public class HelloWorld {
        Run | Debug
2       public static void main(String[] args) {
3           System.out.println("Hello world!");
4           double salary = 18.5;
5           double weeklySalary = salary * 20;
6           System.out.println("Salary " + salary);
7           System.out.println("Weekly Salary " + weeklySalary);
8       }
9   }
```

CS 1 23fa Student Art!

Now we can better understand this :)
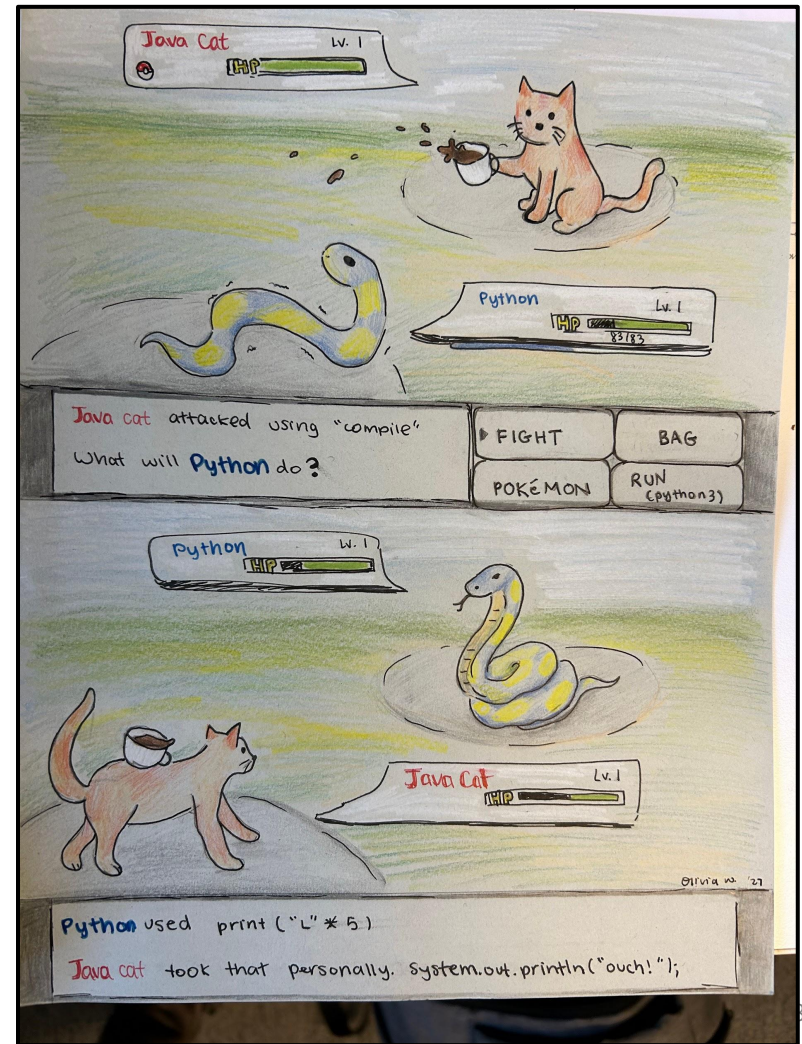
Continued Wednesday

More Java syntax (**`Lec22MoreJava.java`**)

Methods in Java (**`Lec22JavaMethodExample.java`**)
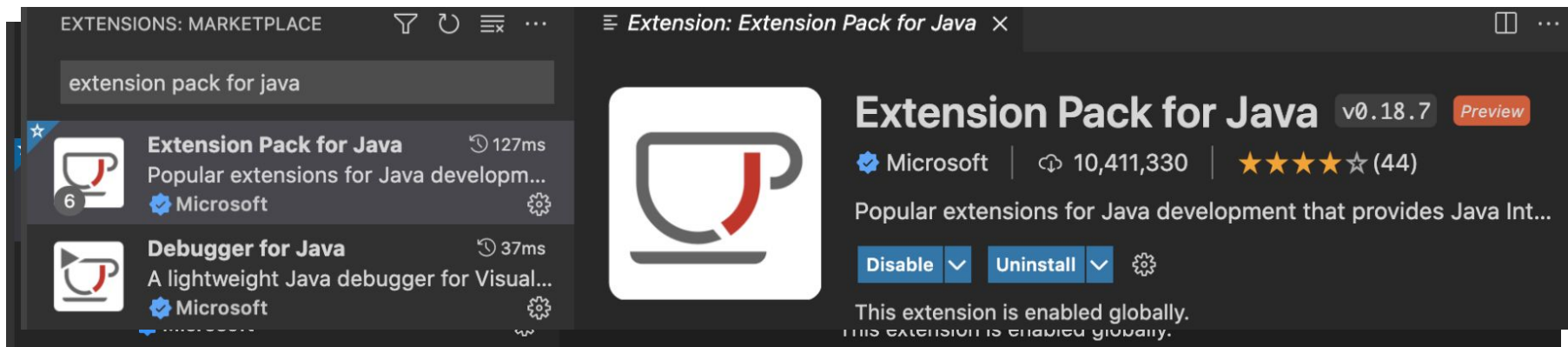
Classes vs. Client Programs in Python vs. Java
- **`Dog.java`** vs. **`Dog.py`**
- **`DogClient.java`** vs. **`dog_client.py`**

VSCode Extension Pack for Java

The Extension Pack for Java in VSCode offers a variety of very useful features to help you when programming in Java (a linter to catch style/syntax errors, a debugger, etc.)

We highly recommend it!

Tips for MP7

You should not be using any data structures, such as Lists, arrays, or Maps (ask El if you're unsure how to implement something without them!)

Do not use recursion (it would be poor practice for any exercise in MP7)

Make sure to utilize the VSCode debugger (Java)!

Review

Recall the three steps to write and run a Python program (e.g. program.py)

1. Write the .py program in VSCode
2. Save
3. Run with **$ python3 program.py**

What are the 4 steps we learned in Lecture 23 to write and run a Java program (e.g. Program.java)?

1. Write the .java program in VSCode
2. Save
3. Compile with **$ javac Program.java**
4. Run with **$ java Program**

More Syntax: VSCode Demo

- Types and variables
- Strings vs. chars
- Functions
- Loops
- if/else

More about `String` vs. `char`

One of the most common bugs students run into when porting Python to Java relates to the distinction Java makes between a `String` and `char`

The following slide was demo'd in class to discuss some of the differences

Hint on MP7 `DNA.java`: make sure you are not converting back and forth between a ***String*** *and character; it's ok to use* `char ch = s.charAt(i)`, *but don't use* `(String) ch` *after)*

Example

```python
def get_even_letters(s):
    """
    Returns all of the even-indexed characters in a given
    string. For example, 'LrmIsm' for 'Lorem Ipsum'.

    Arguments:
        - s (str): string to extract characters from

    Returns:
        - (str): extracted string
    """
    result = ''
    # range(start, stop, step)
    for i in range(0, len(s), 2):
        result += s[i]
    return result
```

lec22_python_fns.py

```java
/**
 * Returns all of the even-indexed characters in a given
 * string. For example, "LrmIsm" for "Lorem Ipsum".
 *
 * @param s - string to extract characters from
 * @return - extracted string
 */
public static String getEvenLetters(String s) {
    String result = "";
    // for (int i = start; i < stop; i += step)
    for (int i = 0; i < s.length(); i += 2) {
        char ch = s.charAt(i);
        result += ch;
    }
    return result;
}
```

Lec22.java

Concatenation with +

Python:

```python
s = ''
s += 'hello'
s += ' '
ch = 'a'
# ch is just a single-character str
# can add a character to string
s += ch   # 'hello a'
# can add a string to a character
ch += s   # 'ahello a'
```

Java:

```java
String s = "";
s += "hello";
s += ' ';
char ch = 'a';
s += ch; // "hello"
// ch += s;
// (compiler error: can only do String += char)
```

We still use **+/+=** to concatenate **String**s in Java, and can concatenate a **char** to a **String**

However, we cannot update a declared **char** variable to concatenate a **String** to it (it would no longer be a **char**!)

Letter-case Methods in Java vs. Python

Python:

```python
# s == 'hello a'
s = s.upper() # 'HELLO A'
s = s.lower() # 'hello a'
# ch == 'a'
ch = ch.upper() # 'A'


ch.isupper() # True
ch.islower() # False
```

Java:

```java
// s == "hello a";
s = s.toUpperCase(); // "HELLO A"
s = s.toLowerCase(); // "hello a"
// ch == 'a'
ch = Character.toUpperCase(ch); // 'A'
ch = Character.toLowerCase(ch); // 'a'
// ch.isUpperCase(); // error
Character.isUpperCase(ch); // false
Character.isLowerCase(ch); // true
```

Here, we see another example of the distinction Java enforces between `String` and `char`

Remember that `char` is primitive, so we use the `Character` wrapper class to access convenient methods like `Character.toUpperCase(char)` -> `boolean`

Strings: Indexing and Characters

Python:

```python
first = s[0]
second = s[2]

last = s[-1]
last = s[len(s) - 1] # same

source = 'UAAUGGAUG'
first_index = source.index('A')
first_index = source.index('a')
# ValueError if 'a' not found!
# we use `in`` instead in Python
has_little_a = 'a' in source
start_index = source.index(START_CODON)
```

Java:

```java
char first = s.charAt(0);
char second = s.charAt(2);
// Can't use -1 indices in Java!
// char last = s.charAt(-1); // error
char last = s.charAt(s.length() - 1);

String source = "UAAUGGAUG";
// Can use String.indexOf(char)
int firstIndex = source.indexOf('A');
firstIndex = source.indexOf('a');
// -1 if 'a' not found
boolean hasLittleA = (firstIndex != -1);
// Can also use String.indexOf(String)
int startIndex = source.indexOf(START_CODON);
// -1 if not found, otherwise is index of substring start
```

Strings: Slicing vs. Substrings

Python:

```python
# "Splicing"
s = 'Hello world!'
sub = s[2:5]  # 'llo'

first = s[0] # 'H'
rest = s[1:] # same as s[1:len(s)]
# 'ello world!'
```

Java:

```java
String s = "Hello world!"
String sub = s.substring(2, 5);
// "llo"
char first = s.charAt(0); // 'H'
// same as s.substring(1, s.length());
String rest = s.substring(1);
// "ello world!"
```

Conditionals and Booleans in Java

In Java, we use `||` and `&&` instead of Python's `or` and `and`, respectively

```java
// Conditionals in Python vs. Java
int n = 5;
// is_digit = (n >= 1 and n <= 10)
boolean isDigit = (n >= 1 && n <= 10);
// true
// is_even_or_negative = (n % 2 == 0 or n < 0)
boolean isEvenOrNegative = (n % 2 == 0 || n < 0);
```

if/elif/else vs. if/else if /else

In Python:

```python
if cond1:
    ...
elif cond2:
    ...
else:
    ...
```

In Java:

```java
if (cond1) {
    ...
} else if (cond2) {
    ...
} else {
    ...
}
```

if/elif/else vs. if/else if /else

In Python:

```python
def aqi_demo():
    aqi_str = '151'
    aqi = int(aqi_str)
    if aqi == 150:
        print("The AQI today is 150. Take caution!")
    elif aqi < 150:
        print("It's healthy to go out outside!")
    else:
        print("It's unhealthy to go outside!")
```

In Java:

```java
public static void aqiDemo() {
    String aqiStr = "151";
    int aqi = Integer.parseInt(aqiStr);
    if (aqi == 150) {
        System.out.println("The AQI today is 150. " +
                            "Take caution!");
    } else if (aqi < 150) {
        System.out.println("It's healthy to go out outside!");
    } else {
        System.out.println("It's unhealthy to go outside!");
    }
}
```

Conditionals and Booleans in Java

Another example with characters (hint: MP7!)

```python
def is_vowel(ch):
    ch = ch.lower()
    return (ch == 'a' or ch == 'e' or ch == 'i' or
            ch == 'o' or ch == 'u')
```

```java
public static boolean isVowel(char ch) {
    ch = Character.toLowerCase(ch);
    return (ch == 'a' || ch == 'e' || ch == 'i' ||
            ch == 'o' || ch == 'u');
}
```

Conditionals and Booleans in Java

Combining a `for` loop, helper method, and `if` statement in Java (javadoc omitted):

```java
public static boolean isVowel(char ch) {
    ch = Character.toLowerCase(ch);
    return (ch == 'a' || ch == 'e' || ch == 'i' ||
            ch == 'o' || ch == 'u');
}

public static int vowelCount(String s) {
    s = s.toLowerCase();
    int count = 0;
    for (int i = 0; i < s.length(); i++) {
        char ch = s.charAt(i);
        if (isVowel(ch)) {
            // equivalent to += 1 in Python
            count++;
        }
    }
    return count;
}
```

Common Bugs: Indentation vs. `{ }`

Python is sensitive to indentation; every `for`/`if`/function block ending with `:` has a body defined by all statements indented within the block, which ends as soon as a statement is de-indented

Java is not sensitive to indentation; blocks are always defined within `{ }` braces, but you should still use indentation within blocks to keep things readable and avoid subtle bugs!

Common Bugs: Indentation vs. { }


```
/**
 * Retu                 ◦ s - string to extract characters from
 * stri    • Returns:
 *                      ◦ - extracted string
 * @par    This method must return a result of type
 * @ret    String Java(603979884)
 */
         View Problem (⌥F8)    No quick fixes available
public static String getEvenLetters(String s) {
    String result = "";
    // for (int i = start; i < stop; i += step)
    for (int i = 0; i < s.length(); i += 2) {
        char ch = s.charAt(i);
        result += ch;
    return result;
    }
}
```

An example; the `return` occurs after the first iteration, but Java requires a **guaranteed String return** (if the loop doesn't enter due to a `String` of length <= 1, nothing is returned, which is a **compiler error**)

With the VSCode Java extension, you can hover over the red lines before compiling to see if it notices the error in advance

```
mehovik@Els-MacBook-Pro:~/eipsum.github.io/cs1/lectures/lec24$ javac Lec24.java
Lec24.java:99: error: missing return statement
        }
        ^
1 error
mehovik@Els-MacBook-Pro:~/eipsum.github.io/cs1/lectures/lec24$
```

Common Bugs: Indentation vs. { }

```java
public static String getEvenLetters(String s) {
    String result = "";
    // for (int i = start; i < stop; i += step)
    for (int i = 0; i < s.length(); i += 2) {
    char ch = s.charAt(i);
        result += ch; }
        return result;
}
```

Because Java executes blocks based on **{** and **}** contents, this *will* compile, but **you should not do this!** You are expected to use the same indentation conventions we've been using 1.) to avoid subtle bugs and 2.) this is very difficult to read, whether you're debugging or others are working on code with you. Consistent indentation will continue to be expected in MP7!

Back to Functions (Python Week 1)

A function is like a machine to perform tasks and possibly return some result

Every function has:

- Behavior (body)
- Parameters (optional)
- Return value (optional)

**Input
(parameters)**

**Output
(return value)**

Function Behavior
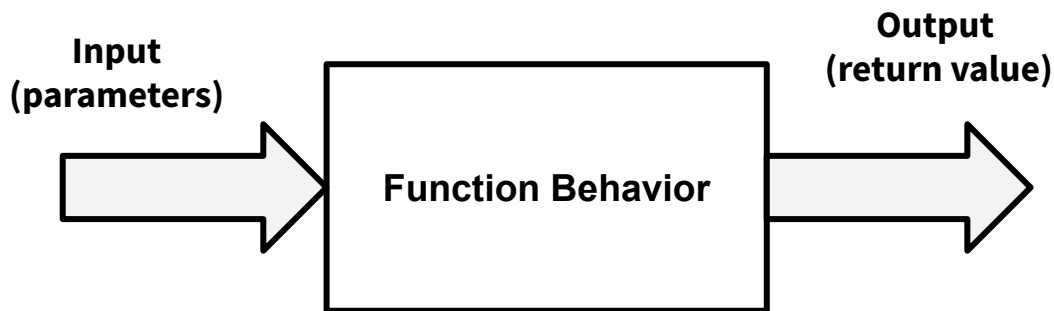
Example with built-in `min` function:

42 → x

min(x, y) → 42

48 → y
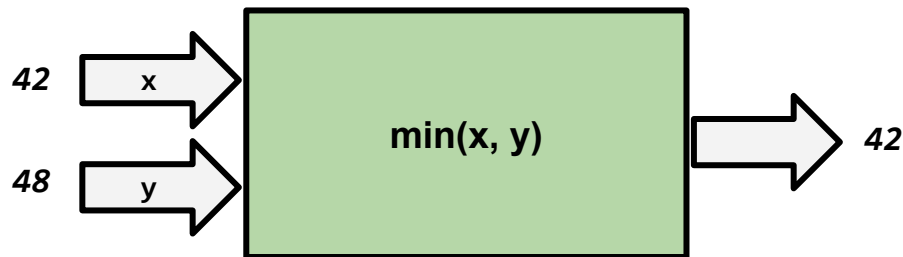
Defining and Calling Functions

Functions may have parameters passed to help generalize functionality and may also specify a return value with the return keyword (**None** if no return specified)

Definition Syntax:

```
def name(<parameters>):
    <body>
    return <value> # optional
```

Definition Examples:

```
def say_hello(name):
    print('Hello ' + name + '!')

def f(x, y):
    return x + 2 * y
```

Function Call Examples:

```
say_hello('world')   # Hello world!
say_hello('Caltech') # Hello Caltech!
ans = f(2, 20)       # ans == 42
```

Defining and Calling ~~Functions~~ Methods

Java methods may have **typed** parameters passed to help generalize functionality and **must** also specify a return value with the return (unless declared a "**void**" method)

Definition Syntax:

```
public static <retType> name(<parameters>):
    <body>
    return <value>; // if not void
```

Note: If a method is defined within a non-executable Class (one defined with a main method), then static is omitted.

Definition Examples:

```
public static void sayHello(String name) {
    System.out.println("Hello " + name + "!");
}


public static int f(int x, int y) {
    return x + 2 * y;
}
```

Method Call Examples:

```
sayHello("world");   // Hello world!
sayHello("Caltech")  // Hello Caltech!
int ans = f(2, 20);  // ans == 42
```

Docstrings vs. javadoc

```python
def vowel_count(s):
    """
    Returns the number of vowels in s, ignoring letter-casing.

    Arguments;
        s (str): string of vowels to count

    Returns:
        (int): count of 'a', 'e', 'i', 'o', 'u' (ignoring casing)
    """
```

```java
/**
 * Returns the number of vowels in s, ignoring letter-casing.
 *
 * @param s - String for counting vowels
 * @return - count of 'a', 'e', 'i', 'o', 'u' (ignoring casing)
 */
public static int vowelCount(String s) {
```

Programs vs. Classes

In MP 6 (and 7) you implement both a client program and 2 classes.

Anything that is runnable (usually with `if __name__ == '__main__'`) is a client program/application in Python

We usually separate client programs from files defining classes for abstraction/good program decomposition (we generally want our classes to be generalized enough to be usable in different client applications)

We'll see this in Java too

This week's lecture code has an *executable* Java program defined with a `main` method (`HelloWorld.java` and `DogClient.java`); `Dog.java` is *not* an executable class.

Python Class vs. Java Class

```python
"""
Program docstring
"""
class ClassName:
    """ class docstring """

    def __init__(self, some_field):
        """ method docstring """
        self.some_field = some_field
```

```java
/**
 * Class (file) javadoc
 */
public class ClassName {
    private int someField;

    /**
     * Constructor/method javadoc
     */
    public ClassName(int someField) {
        this.someField = someField;
    }
}
```

Anything in **black** is a keyword or language-specific token in Python/Java

Green represents the state (attribute/field)

Purple represents the specific class name syntax

Python Class vs. Java Class: `Dog.py`

```python
""" file docstring omitted """
class Dog:
    """ class docstring omitted """

    def __init__(self, fullname, breed, lvl, is_fed=False):
        """ docstring omitted """
        self.name = fullname
        self.breed = breed
        self.good_boi_lvl = lvl
        self.is_fed = is_fed
```

Python Class vs. Java Class: `Dog.java`

```java
/** File javadoc omitted */
public class Dog {
    private String name;      // Name of dog
    private String breed;     // Breed of dog
    private int goodBoiLvl;   // "level" of goodness, always between [0, 10]
    private boolean isFed;    // whether the dog is fed or not

    /** javadoc omitted */
    public Dog(String name, String breed, int goodBoiLevel) {
        this.name = name;
        this.breed = breed;
        this.goodBoiLvl = goodBoiLevel;
        this.isFed = false;
    }
}
```

Python Application vs. Java Application

```python
""" client program docstring omitted """

from Dog import Dog   # Dog is a class in Dog.py

if __name__ == '__main__':
    lorem = Dog('Lorem', 'Boxer/Lab Mix', 4)
    print(lorem.get_name())
    lorem.set_name('Punk')
    print(lorem.get_name())
```

```java
/**
 * Client program javadoc omitted
 */
public class DogClient {
    // No fields for a client class with main!

    public static void main(String[] args) {
        Dog lorem = new Dog("Lorem", "Boxer/Lab Mix", 4);
        System.out.println(lorem.getName());
        lorem.setName("Punk");
        System.out.println(lorem.getName());
    }
}
```

```
mehovik@Els-MacBook-Pro:~/eipsum.github.io/cs1/lectures/lec24$ python3 dog_client.py
Lorem
Punk
mehovik@Els-MacBook-Pro:~/eipsum.github.io/cs1/lectures/lec24$ javac Dog.java
mehovik@Els-MacBook-Pro:~/eipsum.github.io/cs1/lectures/lec24$ javac DogClient.java
mehovik@Els-MacBook-Pro:~/eipsum.github.io/cs1/lectures/lec24$ java DogClient
Lorem
Punk
```

(Basic) Error-Handling in Java

The equivalent of `raise` in Python is `throws` in Java

To throw an exception when given invalid arguments, we use:

`throw new IllegalArgumentException(errMsg);`

You will need to use this a few times in MP7!

```java
 * ...
 * @param lvl — "good boi level" for the new dog, between 0 and 10.
 * @throws IllegalArgumentException if lvl is less than 0 or greater
 *                                  than 10.
 */
public Dog(String fullname, String breed, int lvl) {
    // Best practice to handle exceptions as early as possible.
    if (lvl < 0 || lvl > 10) {
        String errMsg = "Invalid lvl, must be between 0 and 10";
        // Equivalent to:
        // raise ValueError(err_msg)
        throw new IllegalArgumentException(errMsg);
    }
    // Then, continue normal method behavior.
    // refer to fields as this, not self.
    this.name = fullname;
    this.breed = breed;
    this.goodBoiLvl = lvl;
    this.isFed = false;  // lowercase false, not False, in Java
}
```

`Dog.java`

(Basic) Error-Handling in Java

The equivalent of `raise` in Python is `throws` in Java

To throw an exception when given invalid arguments, we use:

`throw new IllegalArgumentException(errMsg);`

You will need to use this a few times in MP7!

```java
 * ...
 * @param lvl - "good boi level" for the new dog, between 0 and 10.
 * @throws IllegalArgumentException if lvl is less than 0 or greater
 *                                  than 10.
 */
public Dog(String fullname, String breed, int lvl) {
    // Best practice to handle exceptions as early as possible.
    if (lvl < 0 || lvl > 10) {
        String errMsg = "Invalid lvl, must be between 0 and 10";
        // Equivalent to:
        // raise ValueError(err_msg)
        throw new IllegalArgumentException(errMsg);
    }
    // Then, continue normal method behavior.
    // refer to fields as this, not self.
    this.name = fullname;
    this.breed = breed;
    this.goodBoiLvl = lvl;
    this.isFed = false;  // lowercase false, not False, in Java
}
```

`Dog.java`

Basic File IO in Python vs. Java

VSCode Demo (AQIs from Lecture 08)

Extra Material: Random in Java

To work with random numbers in Java, we use the **Random** object (requiring import **java.util.Random** at the top)

The two methods that are most commonly used are **r.nextInt(start, stop)** and **r.nextDouble()** (returns a random **double** between **0.0** and **1.0**)

```java
public static void randomDemo() {
    // in B.6., Random mutator = new Random();
    Random r = new Random();
    // Set a rate of success for a random coin flip
    double HEADS_RATE = 0.5;
    int heads = 0;
    // random digit between 0 and 9
    int randDigit = r.nextInt(10);
    int flips = randDigit;
    System.out.println("Flipping " + flips + " coins...");
    for (int i = 0; i < flips; i++) {
        // r.nextDouble() returns a random double between 0.0 and 1.0
        double coinFlip = r.nextDouble();
        // 50% chance of heads
        if (coinFlip < HEADS_RATE) {
            heads++;
        }
    }
    System.out.println(heads + " flips were heads!");

    // Some other examples
    String s = "hello world!";
    // random char index between 0 and length of string
    int randomIndex = r.nextInt(s.length());
    // access the random character by index
    char randomChar = s.charAt(randomIndex);
    System.out.println("Randomly selected " + randomChar +
                       " at index " + randomIndex + " of " + s);
}
```

```
mehovik@Els-MacBook-Pro:~/eipsum.github.io/cs1/lectures/lec25$ javac Lec25.java
mehovik@Els-MacBook-Pro:~/eipsum.github.io/cs1/lectures/lec25$ java Lec25
Flipping 6 coins...
2 flips were heads!
Randomly selected   at index 5 of hello world!
mehovik@Els-MacBook-Pro:~/eipsum.github.io/cs1/lectures/lec25$ javac Lec25.java
mehovik@Els-MacBook-Pro:~/eipsum.github.io/cs1/lectures/lec25$ java Lec25
Flipping 7 coins...
4 flips were heads!
Randomly selected h at index 0 of hello world!
mehovik@Els-MacBook-Pro:~/eipsum.github.io/cs1/lectures/lec25$ []
```