Processing Syntax

(It’s Java syntax)

Processing *is* Java

- Processing is a library of Java methods for creating art
- The syntax is 100% Java syntax
- The overall organization of a program is simpler than that of a Java program
- Processing uses its own IDE

Syntax: Comments

- Single line comments
  - Python: `# Up to the end of the line`
  - Processing: `// Up to the end of the line`
- Multiple line comments
  - Processing:
```
/* This kind of comment can extend over as
   many lines as you like. */
```

- Documentation (doc) comments
  - Python: A string, just inside the function definition
    -
```
def turn_around(direction):
    """Returns the reverse direction"""
```
  - Processing: Just before the method definition
    -
```
/** Returns the reverse direction
    (This can be a multiline comment) */
void turnAround(direction) {
```

Syntax: Lines and semicolons

- In Python,
  - Normally, each statement goes on a separate line
  - Lines ***may*** end with a semicolon
  - Multiple statements may be put on a single line, if they are separated by semicolons

- In Java (and therefore in Processing),
  - Normally, each statement goes on a separate line
  - Lines ***must*** end with a semicolon
  - Multiple statements may be put on a single line, if they are separated by semicolons

Syntax: Variables

- In Python, variables don't have to be declared before they are used, and may hold any kind of value
- In Java (Processing), the type of variables must be declared before they are used, and can hold only values of that type
- Python:
  - `direction = 2`
- Java:
  - ```
    int direction;
    direction = 2;
    ```
  - or
    ```
    int direction = 2;
    ```
- In Python, multiword variable use underscores: `best_score`
- In Java, multiword variables use "camel case": `bestScore`
    ```

Syntax: Simple types

- Integers are declared with `int`
  - For example, `int count;`
- Floating point numbers are declared as `float` or `double`
  - In Java, you should prefer `double`
  - In Processing, you should always use `float`, because that's what the library functions expect
- Logical values are declared as `boolean`
  - In Python, logical values are `True` and `False`
  - In Java, logical values are `true` and `false`
- Strings are declared as `String`
  - Strings are always enclosed in *double* quotes (`" ... "`)

Syntax: Arrays

- Arrays in Java are like lists in Python, except that they are created with a fixed size
  - `int[] scores = new int[40];`
  - or
    `int[] scores;`
    `scores = new int[40];`
- The “first” location in the above array is `scores[0]`, and the last location is `scores[39]`

Syntax: Arithmetic

- Arithmetic in Java is practically the same as in Python
  - `+`, `-`, `*`, and `%` are the same
  - Applied to two integers, `/` gives an integer result
  - Java does not have `**` as an operator
    - Instead of `2**3`, say `Math.pow(2, 3)`
  - Parentheses are used the same as they are in Python
- You can use an `int` where a `float` or `double` is expected, but you can't use a `float` or `double` where an `int` is expected
- Processing has several predefined variables: `width` and `height` (of the window), `PI`, etc.

Syntax: Logic

- Logical variables are declared as `boolean`, and may have the value `true` or `false` (not capitalized!)
  - "And" is the binary operator `&&`
  - "Or" is the binary operator `||`
  - "Exclusive or" is the binary operator `^`
  - "Not" is the prefix operator `!`
- Numbers may be compared with any of
  `<` `<=` `==` `!=` `>=` `>`
  and the result will be a `boolean`
- To compare strings, use ***string1*`.equals(`*string2*`)`**
  - **Do not use *string1* `==` *string2*** -- sometimes it works, sometimes it doesn't!

Syntax: `if` statements

- Python:

```
if column == -1:
  return move('R')
elif column == 8:
  return move('L')
elif row == -1:
  return move('D')
elif row == 8:
  return move('U')
else:
  print("Error!")
  return 0
```

- The same thing in Java:

```
if (column == -1) {
    return move("R");
} else if (column == 8) {
    return move("L");
} else if (row == -1) {
    return move("D");
} else if (row == 8) {
    return move("U");
} else {
    print("Error!");
    return 0;
}
```

- Conditions must be in parentheses
- There is no `elif`
- Grouping is done with braces, *not* colon (`:`)
- Strings must be in double quotes
- Every statement ends with a semicolon (`;`)

Syntax: `while` loops

- Python:
- ```
  n = 1
  while n < 1000:
    n = 2 * n
  ```

- The same thing in Java:
- ```
  n = 1;
  while (n < 1000) {
      n = 2 * n;
  }
  ```

- `n` must have been previously declared
- Conditions must be in parentheses
- Grouping is done with braces, *not* colon (`:`)
- Every statement ends with a semicolon (`;`)

Syntax: `for` loops

- Python:

```
primes = [2, 3, 5, 7, 11]
sum = 0
for p in primes:
  sum += p
```

- or

```
primes = [2, 3, 5, 7, 11]
sum = 0
for i in range(0, len(primes)):
  sum += primes[i]
```

- The same thing in Java:

```
int[] primes = {2, 3, 5, 7, 11};
int sum = 0;
for (p : primes) {
  sum += p;
}
```

- or

```
int[] primes = {2, 3, 5, 7, 11};
int sum = 0;
for (i = 0; i < primes.length; i++) {
  sum += primes[i];
}
```

- The parts of the second `for` loop are
  1. The initialization of a loop variable
  2. The test for remaining in the loop
  3. The modification of the loop variable

Syntax: methods (“functions”)

- Python:

```
def intlog(n):
  if n == 1:
    return 1
  else:
    return 1 + intlog(n // 2)
```

- The same thing in Java:

```
int intlog(int n) {
    if (n == 1) {
        return 1;
    } else {
        return 1 + intlog(n / 2);
    }
}
```

- The method starts by specifying the type of value to be returned
- The type of every argument is specified
- The entire method body is enclosed in braces, `{ }`
- Recursion is fully supported
- Integer division is `/`, not `//`

The End