# FUNCTION OBJECTS

Instructors: Crista Lopes

# How to model functions as data in OOP?

```
def read_file(path_to_file, func):
    with  open(path_to_file) as f:
        data = f.read()
    func(data, normalize)

def filter_chars(str_data, func):
    pattern = re.compile('[\W_]+')
    func(pattern.sub(' ', str_data), scan)

def normalize(str_data, func):
    func(str_data.lower(), remove_stop_words)

def scan(str_data, func):
    func(str_data.split(), frequencies)

def remove_stop_words(word_list, func):
    with open('../stop_words.txt') as f:
        stop_words = f.read().split(',')
    # add single-letter words
    stop_words.extend(list(string.ascii_lowercase))
    func([w for w in word_list if not w in stop_words], sort)

def frequencies(word_list, func):
    wf = {}
    for w in word_list:
        if w in wf:
            wf[w] += 1
        else:
            wf[w] = 1
    func(wf, print_text)

def sort(wf, func):
    func(sorted(wf.iteritems(), key=operator.itemgetter(1),
        reverse=True), no_op)

def print_text(word_freqs, func):
    for (w, c) in word_freqs[0:25]:
        print w, "-", c
    func(None)

def no_op(func):
    return

#
# The main function
#
read_file(sys.argv[1], filter_chars)
```

```
TFTheOne(sys.argv[1])\
.bind(read_file)\
.bind(filter_chars)\
.bind(normalize)\
.bind(scan)\
.bind(remove_stop_words)\
.bind(frequencies)\
.bind(sort)\
.bind(top25_freqs)\
.printme()
```

# First attempt

```
class ReadFile {
  String readFile(string arg)
  {
    string contents = // Code for reading a file
    return contents;
  }
}
```

and then

```
new ReadFile().readFile(...);
```

or

```
ReadFile rf = new ReadFile();
someObj.meth(rf);
```

# First attempt

```
class FilterChars
{
  String filter(String arg)
  {
    string fcontents = // Code for filtering
    return fcontents;
  }
}
```

same

# Problem: no commonality between function classes

```
class TFTheOne
{
  private Object value;
  TFTheOne(Object v) { value = v; }

  public TFTheOne bind(??? func) {
    value = func.???(value);
    return this;
  }

  public printme() {
    System.out.println(value);
  }
}
```

Object?

➔ Giant switch statement

# Problem: no commonality between function classes

```
class TFTheOne
{
  private Object value;
  TFTheOne(Object v) { value = v; }

  public TFTheOne bind(Object func) {
    if (func instanceof ReadFile)
       value = func.ReadFile(value);
    else if (func instanceof FilterChars)
       value = func.FilterChars(value);
    else if ...

    return this;
  }
}
```

➔ Giant switch statement

# A Function Object Interface

```
interface IFunction
{
    Object call(Object arg);
}
```

More variations are possible...

# The One – Function examples

```
class ReadFile implements IFunction
{
  Object call(Object arg)
  {
    string fileName = (String)arg;
    string contents = // Code for reading a file
    return contents;
  }
}

 class FilterChars implements IFunction
 {
   Object call(Object arg)
   {
     string data = (String)arg;
     string fcontents = // Code for filtering
     return fcontents;
   }
 }
```

# The One

```
class TFTheOne
{
  private Object value;
  TFTheOne(Object v) { value = v; }

  public TFTheOne bind(IFunction func) {
    value = func.call(value);
    return this;
  }

  public printme() {
    System.out.println(value);
  }
}
```

Beautifully generic!

# The One – Main

```
public static void main(String[] args) {
  TFTheOne one = new TFTheOne(args[...]);

  one.bind(new ReadFile()).bind(new FilterChars()).bind ...
}
```

# Kick Forward (CPS)

```
interface IFunction
{
    void call(Object arg, IFunction func);
}
```

and then

```
class ReadFile implements IFunction
{
  void call(Object arg, IFunction func)
  {
    string fileName = (String)arg;
    string contents = // Code for reading a file
    func(contents, new FilterChars());
  }
}
```

# Functions as Objects

- Define a clean “function” interface
- Define one class per function, each implementing the interface
- When you need to pass the function, instantiate the class and pass that object

# More generally

- Objects are capsules of procedures/functions
  - with or without state
- Functions as data in pure OOP → function objects
- If you need to manipulate them independent of their type → use interface / super