



Exploration Seminar 5

Functional Python

Special thanks to Scott Shawcroft, Ryan Tucker, and Paul Beck for their work on these slides.

Except where otherwise noted, this work is licensed under:

<http://creativecommons.org/licenses/by-nc-sa/3.0>

Redundant Java

```
public void method1(int[] arr) {
    for (int i = 0; i < arr.length; i++) {
        arr[i] = function1(arr[i]);
    }
}

public void method2(int[] arr) {
    for (int i = 0; i < arr.length; i++) {
        arr[i] = function2(arr[i]);
    }
}

public int function1(int x) {
    return x * 2;
}

public int function2(int x) {
    return x + 2;
}
```

Functions as parameters

- Have you ever wanted to pass an entire function as a parameter
- Python has functions as first-class citizens, so you can do this
- You simply pass the functions by name

Properties of Functions

Field	Description
<code>__name__</code>	This is the name of the function. This only have a meaningful value is the function is defined with "def".
<code>__class__</code>	This is a reference to the class a method belongs to.
<code>__code__</code>	This is a reference to the code object used in the implementation of python
<code>__doc__</code>	This is the documentation string for the function.

inspect

- A useful class for inspecting functions and classes.
 - `from inspect import *`

Field	Description
<code>getdoc(x)</code>	Returns a pretty version of the docstring for the give object.
<code>getcomments(x)</code>	Returns the comments that appear just above the given function/class/module.
<code>getsource(x)</code>	Returns the source code for the given function/class/module
<code>getmembers(x)</code>	Returns a list of the members (fields and methods) of a class

Function Parameter Example

ex.py

```
1 def mult_2(x):
2     return x * 2
3
4 def add_2(x):
5     return x + 2
6
7 def opp_on_item(item, func):
8     return func(item)
9
0 #main
1 opp_on_item(12, mult_2)           #result: 24
2 opp_on_item(12, add_2)           #result: 14
3
```

Lambda

- Sometimes you need a simply arithmetic function
- Its silly to write a method for it, but redundant not too
- With lambda we can create quick simple functions
- Facts
 - Lambda functions can only be comprised of a single expression
 - No loops, no calling other methods
 - Lambda functions can take any number of variables

Syntax:

```
lambda param1, ..., paramn : expression
```

Lambda Syntax

lambda.py

```
1 #Example 1
2 square_func = lambda x : x**2
3 square_func(4) #return: 16
4
5 #Example 2
6 close_enough = lambda x, y : abs(x - y) < 3
7 close_enough(2, 4) #return: True
8
9 #Example 3
0 def get_func(n) :
1     return lambda x : x * n + x % n
2 my_func = get_func(13)
3 my_func(4) #return: 56
```


operator

- Most of the built-in functions (`len`, `+`, `*`, `<`) can be accessed through the `operator` module
- Need to import the `operator` module
 - `from operator import *`

Operator	Function
<code>-</code>	<code>neg(x)</code>
<code>+</code>	<code>pos(x)</code>

Operator	Function
<code>-</code>	<code>sub(x, y)</code>
<code>+</code>	<code>add(x, y)</code>
<code>*</code>	<code>__mul__(self, other)</code>

Operator	Function
<code>==</code>	<code>eq(x, y)</code>
<code>!=</code>	<code>ne(x, y)</code>
<code><</code>	<code>lt(x, y)</code>
<code>></code>	<code>gt(x, y)</code>
<code><=</code>	<code>le(x, y)</code>
<code>>=</code>	<code>ge(x, y)</code>

Partially Instantiated Functions

- We have seen that we can create lambda functions for quick functions on the go
- We have also seen that we can use the built in operators through the `operator` class
- What we would like to do is use the built in operators with a silly lambda function
- We can do this by partially instantiating function with the `partial` function from the `functools` package
 - You supply some of the parameters and get a function back the needs the rest of the parameters in order to execute

partial

partial.py

```
1 def mult1(x):
2     return 2 * x
3 mult2 = lambda x : 2 * x
4 mult3 = partial(mul, 2)
5
6 x = 10
7
8 print(mult1(5));           #10
9 print(mult2(5));           #10
0 print(mult3(5));           #10
```

Higher-Order Functions

- A higher-order function is a function that takes another function as a parameter
- They are “higher-order” because it’s a function of a function
- Examples
 - Map
 - Reduce
 - Filter
- Lambda works great as a parameter to higher-order functions if you can deal with its limitations

Transform Example

- Let's write a method called transform that takes a list and a function as parameters and applies the function to each element of the list

transform.py

```
1 def mult_2(x):
2     return x * 2
3 ...
4 #Main
5 x = [1, 2, 3]
6 transform(x, mult_2)
7 print(x)                                #[2, 4, 6]
```

Transform Solution

transform.py

```
1 def transform(arr, func):
2     for i in range(len(arr)):
3         arr[i] = func(arr[i])
4
5 x = [1, 2, 3]
6 transform(x, mult_2)
7 print(x)                #[2, 4, 6]
8
9
```