Python

First-Class Functions
An integer is 32 bits of data...
An integer is 32 bits of data...
...that variables can refer to
An integer is 32 bits of data...
...that variables can refer to
A string is a sequence of bytes representing characters...
An integer is 32 bits of data...
...that variables can refer to

A string is a sequence of bytes representing characters...
...that variables can refer to
An integer is 32 bits of data...
...that variables can refer to

A string is a sequence of bytes representing characters...
...that variables can refer to

A function is a sequence of bytes representing instructions...
An integer is 32 bits of data...
...that variables can refer to
A string is a sequence of bytes representing characters...
...that variables can refer to
A function is a sequence of bytes representing instructions...
...and yes, variables can refer to them to
An integer is 32 bits of data...
...that variables can refer to
A string is a sequence of bytes representing characters...
...that variables can refer to
A function is a sequence of bytes representing instructions...
...and yes, variables can refer to them to
This turns out to be very useful, and very powerful
What happens when a function is defined
What happens when a function is defined

def threshold(signal):
    return 1.0 / sum(signal)
What happens when a function is defined

def threshold(signal):
    return 1.0 / sum(signal)
What happens when a function is defined

```python
def threshold(signal):
    return 1.0 / sum(signal)
```

Not really very different from:

```python
name = 'Alan Turing'
```

Python First-Class Functions
Can assign that value to another variable
Can assign that value to another variable

def threshold(signal):
    return 1.0 / sum(signal)
Can assign that value to another variable

```python
def threshold(signal):
    return 1.0 / sum(signal)

t = threshold
```

First-Class Functions
Can assign that value to another variable

```python
def threshold(signal):
    return 1.0 / sum(signal)

t = threshold
print t([0.1, 0.4, 0.2])
1.42857
```
Can put (a reference to) the function in a list
Can put (a reference to) the function in a list

def area(r):
    return PI * r * r

def circumference(r):
    return 2 * PI * r
Can put (a reference to) the function in a list

def area(r):
    return PI * r * r

def circumference(r):
    return 2 * PI * r

funcs = [area, circumference]

Python

First-Class Functions
Can put (a reference to) the function in a list

def area(r):
    return PI * r * r

def circumference(r):
    return 2 * PI * r

funcs = [area, circumference]

for f in funcs:
    print f(1.0)
Can put (a reference to) the function in a list

```python
def area(r):
    return PI * r * r

def circumference(r):
    return 2 * PI * r

funcs = [area, circumference]

for f in funcs:
    print f(1.0)
```

```
3.14159
6.28318
```
Can pass (a reference to) the function into a function
Can pass (a reference to) the function into a function

```python
def call_it(func, value):
    return func(value)
```
Can pass (a reference to) the function into a function

```python
def call_it(func, value):
    return func(value)

print call_it(area, 1.0)
3.14159
```
Can pass (a reference to) the function into a function

```python
def call_it(func, value):
    return func(value)

print call_it(area, 1.0)
3.14159

print call_it(circumference, 1.0)
6.28318
```
Can now write functions of functions
Can now write functions of functions

```python
def do_all(func, values):
    result = []
    for v in values:
        temp = func(v)
        result.append(temp)
    return result
```

Python

First-Class Functions
Can now write functions of functions

```python
def do_all(func, values):
    result = []
    for v in values:
        temp = func(v)
        result.append(temp)
    return result

print do_all(area, [1.0, 2.0, 3.0])
```

First-Class Functions
Can now write functions of functions

def do_all(func, values):
    result = []
    for v in values:
        temp = func(v)
        result.append(temp)
    return result

print do_all(area, [1.0, 2.0, 3.0])

[3.14159, 12.56636, 28.27431]
Can now write functions of functions

def do_all(func, values):
    result = []
    for v in values:
        temp = func(v)
        result.append(temp)
    return result

print do_all(area, [1.0, 2.0, 3.0])
[3.14159, 12.56636, 28.27431]

def slim(text):
    return text[1:-1]
Can now write functions of functions

def do_all(func, values):
    result = []
    for v in values:
        temp = func(v)
        result.append(temp)
    return result

print do_all(area, [1.0, 2.0, 3.0])
[3.14159, 12.56636, 28.27431]

def slim(text):
    return text[1:-1]

print do_all(slim, ['abc', 'defgh'])
b efg
Higher-order functions allow re-use of control flow
Higher-order functions allow re-use of control flow

```python
def combine_values(func, values):
    current = values[0]
    for i in range(1, len(values)):
        current = func(current, v)
    return current
```
**Higher-order functions** allow re-use of control flow

```python
def combine_values(func, values):
    current = values[0]
    for i in range(1, len(values)):
        current = func(current, v)
    return current

def add(x, y): return x + y
def mul(x, y): return x * y
```
Higher-order functions allow re-use of control flow

```python
def combine_values(func, values):
    current = values[0]
    for i in range(1, len(values)):
        current = func(current, v)
    return current

def add(x, y): return x + y
def mul(x, y): return x * y

print combine_values(add, [1, 3, 5])
9
```
Higher-order functions allow re-use of control flow

def combine_values(func, values):
    current = values[0]
    for i in range(1, len(values)):
        current = func(current, v)
    return current

def add(x, y): return x + y
def mul(x, y): return x * y

print combine_values(add, [1, 3, 5])  
9
print combine_values(mul, [1, 3, 5])  
15
Without higher order functions
Without higher order functions

<table>
<thead>
<tr>
<th></th>
<th>op_1</th>
<th>op_2</th>
<th>op_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>data_structure_A</td>
<td>do_1A</td>
<td>do_2A</td>
<td>do_3A</td>
</tr>
<tr>
<td>data_structure_B</td>
<td>do_1B</td>
<td>do_2B</td>
<td>do_3B</td>
</tr>
<tr>
<td>data_structure_C</td>
<td>do_1C</td>
<td>do_2C</td>
<td>do_3C</td>
</tr>
</tbody>
</table>
Without higher order functions

<table>
<thead>
<tr>
<th></th>
<th>op_1</th>
<th>op_2</th>
<th>op_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>data_structure_A</td>
<td>do_1A</td>
<td>do_2A</td>
<td>do_3A</td>
</tr>
<tr>
<td>data_structure_B</td>
<td>do_1B</td>
<td>do_2B</td>
<td>do_3B</td>
</tr>
<tr>
<td>data_structure_C</td>
<td>do_1C</td>
<td>do_2C</td>
<td>do_3C</td>
</tr>
</tbody>
</table>

total: 9

Python

First-Class Functions
Without higher order functions

<table>
<thead>
<tr>
<th></th>
<th>op_1</th>
<th>op_2</th>
<th>op_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>data_structure_A</td>
<td>do_1A</td>
<td>do_2A</td>
<td>do_3A</td>
</tr>
<tr>
<td>data_structure_B</td>
<td>do_1B</td>
<td>do_2B</td>
<td>do_3B</td>
</tr>
<tr>
<td>data_structure_C</td>
<td>do_1C</td>
<td>do_2C</td>
<td>do_3C</td>
</tr>
</tbody>
</table>

total: 9

With higher order functions

Python

First-Class Functions
Without higher order functions

<table>
<thead>
<tr>
<th></th>
<th>op_1</th>
<th>op_2</th>
<th>op_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>data_structure_A</td>
<td>do_1A</td>
<td>do_2A</td>
<td>do_3A</td>
</tr>
<tr>
<td>data_structure_B</td>
<td>do_1B</td>
<td>do_2B</td>
<td>do_3B</td>
</tr>
<tr>
<td>data_structure_C</td>
<td>do_1C</td>
<td>do_2C</td>
<td>do_3C</td>
</tr>
<tr>
<td><strong>total:</strong></td>
<td><strong>9</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

With higher order functions

<table>
<thead>
<tr>
<th></th>
<th>op_1</th>
<th>op_2</th>
<th>op_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>operate_on_A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>operate_on_B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>operate_on_C</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Python

First-Class Functions
### Without higher order functions

<table>
<thead>
<tr>
<th>data_structure_A</th>
<th>op_1</th>
<th>op_2</th>
<th>op_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>do_1A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>do_2A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>do_3A</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>data_structure_B</th>
<th>op_1</th>
<th>op_2</th>
<th>op_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>do_1B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>do_2B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>do_3B</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>data_structure_C</th>
<th>op_1</th>
<th>op_2</th>
<th>op_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>do_1C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>do_2C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>do_3C</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**total: 9**

### With higher order functions

<table>
<thead>
<tr>
<th>operate_on_A</th>
<th>op_1</th>
<th>op_2</th>
<th>op_3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>operate_on_B</th>
<th>op_1</th>
<th>op_2</th>
<th>op_3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>operate_on_C</th>
<th>op_1</th>
<th>op_2</th>
<th>op_3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**total: 6**

---

**Python**

**First-Class Functions**
Must need to know *something* about the function in order to call it
Must need to know *something* about the function in order to call it
Like number of arguments
Must need to know *something* about the function in order to call it
Like *number of arguments*
Must need to know *something* about the function in order to call it

Like *number of arguments*

def add_all(*args):
    total = 0
    for a in args:
        total += a
    return total
Must need to know *something* about the function in order to call it

Like **number of arguments**

```python
def add_all(*args):
    total = 0
    for a in args:
        total += a
    return total
```
Must need to know *something* about the function in order to call it.

Like number of arguments

```python
def add_all(*args):
    total = 0
    for a in args:
        total += a
    return total

print(add_all())
0
```
Must need to know *something* about the function in order to call it

Like number of arguments

def add_all(*args):
    total = 0
    for a in args:
        total += a
    return total

print add_all()
0
print add_all(1, 2, 3)
6
Combine with "regular" parameters
Combine with "regular" parameters

def combine_values(func, *values):
    current = values[0]
    for i in range(1, len(values)):
        current = func(current, v)
    return current
def combine_values(func, *values):
    current = values[0]
    for i in range(1, len(values)):
        current = func(current, v)
    return current
Combine with "regular" parameters

```python
def combine_values(func, *values):
    current = values[0]
    for i in range(1, len(values)):
        current = func(current, v)
    return current

print combine_values(add, 1, 3, 5)
9
```
Combine with "regular" parameters

def combine_values(func, *values):
    current = values[0]
    for i in range(1, len(values)):
        current = func(current, v)
    return current

print combine_values(add, 1, 3, 5)
9

What does combine_values(add) do?
def combine_values(func, *values):
    current = values[0]
    for i in range(1, len(values)):
        current = func(current, v)
    return current

print combine_values(add, 1, 3, 5)
9

What does combine_values(add) do?
What should it do?
filter(F, S) select elements of S for which F is True
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filter(F, S)</td>
<td>select elements of S for which F is True</td>
</tr>
<tr>
<td>map(F, S)</td>
<td>apply F to each element of S</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>filter(F, S)</td>
<td>select elements of S for which F is True</td>
</tr>
<tr>
<td>map(F, S)</td>
<td>apply F to each element of S</td>
</tr>
<tr>
<td>reduce(F, S)</td>
<td>use F to combine all elements of S</td>
</tr>
</tbody>
</table>

**Python**

First-Class Functions
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>filter(F, S)</code></td>
<td>Select elements of S for which F is True</td>
</tr>
<tr>
<td><code>map(F, S)</code></td>
<td>Apply F to each element of S</td>
</tr>
<tr>
<td><code>reduce(F, S)</code></td>
<td>Use F to combine all elements of S</td>
</tr>
</tbody>
</table>

```python
def positive(x): return x >= 0
print filter(positive, [-3, -2, 0, 1, 2])
[0, 1, 2]
```
<table>
<thead>
<tr>
<th>Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filter(F, S)</td>
<td>select elements of S for which F is True</td>
</tr>
<tr>
<td>map(F, S)</td>
<td>apply F to each element of S</td>
</tr>
<tr>
<td>reduce(F, S)</td>
<td>use F to combine all elements of S</td>
</tr>
</tbody>
</table>

```python
def positive(x): return x >= 0
def negate(x): return -x
def filter(F, S):
    return [s for s in S if F(s)]
def map(F, S):
    return [F(s) for s in S]
def reduce(F, S):
    result = S[0]
    for s in S[1:]:
        result = F(result, s)
    return result
```

```python
print filter(positive, [-3, -2, 0, 1, 2])  # [0, 1, 2]
print map(negate, [-3, -2, 0, 1, 2])       # [3, 2, 0, -1, -2]
```
filter(F, S) | select elements of S for which F is True
---|---
map(F, S) | apply F to each element of S
reduce(F, S) | use F to combine all elements of S

def positive(x): return x >= 0
print filter(positive, [-3, -2, 0, 1, 2])
[0, 1, 2]

def negate(x): return -x
print map(negate, [-3, -2, 0, 1, 2])
[3, 2, 0, -1, -2]

def add(x, y): return x+y
print reduce(add, [-3, -2, 0, 1, 2])
-2
What is programming?
What is programming?

Novice: writing instructions for the computer
What is programming?

Novice: writing instructions for the computer

Expert: creating and combining abstractions
What is programming?

Novice: writing instructions for the computer

Expert: creating and combining abstractions
figure out what the pattern is
What is programming?
Novice: writing instructions for the computer
Expert: creating and combining abstractions
figure out what the pattern is
write it down as clearly as possible
What is programming?

Novice: writing instructions for the computer

Expert: creating and combining abstractions

figure out what the pattern is

write it down as clearly as possible

build more patterns on top of it
What is programming?

Novice: writing instructions for the computer

Expert: creating and combining abstractions
figure out what the pattern is
write it down as clearly as possible
build more patterns on top of it

But limits on short-term memory still apply
What is programming?

Novice: writing instructions for the computer

Expert: creating and combining abstractions

figure out what the pattern is

write it down as clearly as possible

build more patterns on top of it

But limits on short-term memory still apply

Hard to understand what meta-meta-functions actually do