Function
Examples
Class outline:

- *Args
- Currying
- Midterm Review
*Args
The *args syntax

What if you want a function to accept any number of arguments?

The built-in `max` function allows that:

```python
max(1, 2)  # 2
max(10, 30, 20)  # 30
max(-2, 33, -40, 400, 321)  # 400
```
The *args syntax

What if you want a function to accept any number of arguments?

The built-in `max` function allows that:

```python
max(1, 2) # 2
max(10, 30, 20) # 30
max(-2, 33, -40, 400, 321) # 400
```

That's possible by using the `*args` syntax in the function definition.

```python
def max(*args):
    # Do something with *args
```
Forwarding the *args

One way to use *args is to send those arguments into another function.

```python
def min_and_max(*args):
    return min(*args), max(*args)
```

```
min_and_max(-2, 33, -40, 400, 321)
```
Forwarding the *args

One way to use *args is to send those arguments into another function.

```python
def min_and_max(*args):
    return min(*args), max(*args)
```

```python
min_and_max(-2, 33, -40, 400, 321) # -40, 400
```
Forwarding HOF example

A HOF can return a function that can be called with any number of arguments, and then forward those arguments inside the returned function.

def printed(f):
    def print_and_return(*args):
        result = f(*args)
        print('Result:', result)
        return result

    return print_and_return

printed_max = printed(max)
printed_max(-2, 33, -40, 400, 321)
Currying
(Reminder) Function currying

**Currying:** Converting a function that takes multiple arguments into a single-argument higher-order function.

A function that currys any two-argument function:

```python
def curry2(f):
    def g(x):
        def h(y):
            return f(x, y)
        return h
    return g
```

```python
from operator import add

make_adder = curry2(add)
make_adder(2)(3)
```

```python
curry2 = lambda f: lambda x: lambda y: f(x, y)
```
Use case for currying #1

Whenever another function requires a function that only takes one argument:

```python
def transform_numbers(num1, num2, num3, transform):
    return transform(num1), transform(num2), transform(num3)

transform_numbers(3, 4, 5, curry2(add)(60))
```
Use case for currying #1

Whenever another function requires a function that only takes one argument:

```python
def transform_numbers(num1, num2, num3, transform):
    return transform(num1), transform(num2), transform(num3)

transform_numbers(3, 4, 5, curry2(add)(60))
```

Alternate approach:

```python
transform_numbers(3, 4, 5, lambda x: add(60, x))
```
Use case for currying #2

Turning a generalized function into a specialized function:

```python
def html_tag(tag_name, text):
    return "<" + tag_name + ">" + text + "</" + tag_name + ">"

p_tag = curry2(html_tag)("p")
p_tag("hello hello")
```
Use case for currying #2

Turning a generalized function into a specialized function:

```python
def html_tag(tag_name, text):
    return "<" + tag_name + ">" + text + "</" + tag_name + ">

p_tag = curry2(html_tag)("p")
p_tag("hello hello")
```

Alternate approach:

```python
import functools

p_tag = functools.partial(html_tag, "p")
p_tag("hello hello")
```
Why learn currying in Python?

It's good for you!

CS61A introduces many concepts that aren't standard Python practice, but that show up in other languages. Currying is a very common practice in functional programming languages like Haskell or Clojure.
Review
What Would Python Do? #1

WWPD exercises test our understanding of how Python evaluates code and what it chooses to display in the shell.

<table>
<thead>
<tr>
<th>The expression</th>
<th>Evaluates to</th>
<th>Interactive output</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>print(5)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>print(print(5))</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**What Would Python Do? #1**

WWPD exercises test our understanding of how Python evaluates code and what it chooses to display in the shell.

<table>
<thead>
<tr>
<th>The expression</th>
<th>Evaluates to</th>
<th>Interactive output</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td><code>print(5)</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>print(print(5))</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What Would Python Do? #1

WWPD exercises test our understanding of how Python evaluates code and what it chooses to display in the shell.

<table>
<thead>
<tr>
<th>The expression</th>
<th>Evaluates to</th>
<th>Interactive output</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>print(5)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>print(print(5))</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What Would Python Do? #1

WWPD exercises test our understanding of how Python evaluates code and what it chooses to display in the shell.

<table>
<thead>
<tr>
<th>The expression</th>
<th>Evaluates to</th>
<th>Interactive output</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>print(5)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>print(print(5))</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```python
>>> 5
5
```
What Would Python Do? #1

WWPD exercises test our understanding of how Python evaluates code and what it chooses to display in the shell.

<table>
<thead>
<tr>
<th>The expression</th>
<th>Evaluates to</th>
<th>Interactive output</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td><code>print(5)</code></td>
<td>None</td>
<td></td>
</tr>
<tr>
<td><code>print(print(5))</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```python
>>> 5
5
```
What Would Python Do? #1

WWPD exercises test our understanding of how Python evaluates code and what it chooses to display in the shell.

<table>
<thead>
<tr>
<th>The expression</th>
<th>Evaluates to</th>
<th>Interactive output</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>print(5)</td>
<td>None</td>
<td>5</td>
</tr>
<tr>
<td>print(print(5))</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```python
c> 5
5
```
What Would Python Do? #1

WWPD exercises test our understanding of how Python evaluates code and what it chooses to display in the shell.

<table>
<thead>
<tr>
<th>The expression</th>
<th>Evaluates to</th>
<th>Interactive output</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>print(5)</td>
<td>None</td>
<td>5</td>
</tr>
<tr>
<td>print(print(5))</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```python
>>> 5
5
>>> print(5)
5
```
What Would Python Do? #1

WWPD exercises test our understanding of how Python evaluates code and what it chooses to display in the shell.

<table>
<thead>
<tr>
<th>The expression</th>
<th>Evaluates to</th>
<th>Interactive output</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>print(5)</td>
<td>None</td>
<td>5</td>
</tr>
<tr>
<td>print(print(5))</td>
<td>None</td>
<td></td>
</tr>
</tbody>
</table>

```python
g 5
5
>>> 5
5
>>> print(5)
5
```
What Would Python Do? #1

WWPD exercises test our understanding of how Python evaluates code and what it chooses to display in the shell.

<table>
<thead>
<tr>
<th>The expression</th>
<th>Evaluates to</th>
<th>Interactive output</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>print(5)</td>
<td>None</td>
<td>5</td>
</tr>
<tr>
<td>print(print(5))</td>
<td>None</td>
<td>5 None</td>
</tr>
</tbody>
</table>

```python
>>> 5
5
>>> print(5)
5
```
What Would Python Do? #1

WWPD exercises test our understanding of how Python evaluates code and what it chooses to display in the shell.

<table>
<thead>
<tr>
<th>The expression</th>
<th>Evaluates to</th>
<th>Interactive output</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>print(5)</td>
<td>None</td>
<td>5</td>
</tr>
<tr>
<td>print(print(5))</td>
<td>None</td>
<td>5</td>
</tr>
</tbody>
</table>

>> 5
5
>>> print(5)
5
>>> print(print(5))
5
None
What Would Python Do? #2

```
def delay(arg):
    print('delayed')
def g():
    return arg
return g
```

<table>
<thead>
<tr>
<th>The expression</th>
<th>Evaluates to</th>
<th>Interactive output</th>
</tr>
</thead>
<tbody>
<tr>
<td>delay(6)()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>delay(delay)()(6)()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>print(delay(print)())(4))</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## What Would Python Do? #2

```python
def delay(arg):
    print('delayed')
    def g():
        return arg
    return g
```

<table>
<thead>
<tr>
<th>The expression</th>
<th>Evaluates to</th>
<th>Interactive output</th>
</tr>
</thead>
<tbody>
<tr>
<td>delay(6)()</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>delay(delay)()()()6)()()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>print(delay(print)()()4))</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What Would Python Do? #2

```python
def delay(arg):
    print('delayed')
    return arg

def g():
    return arg
return g
```

<table>
<thead>
<tr>
<th>The expression</th>
<th>Evaluates to</th>
<th>Interactive output</th>
</tr>
</thead>
<tbody>
<tr>
<td>delay(6)()</td>
<td>6</td>
<td>delayed</td>
</tr>
<tr>
<td>delay(delay)()(6)()</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>print(delay(print)()(4))</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
def delay(arg):
    print('delayed')
    def g():
        return arg
    return g

def g():
    return arg

<table>
<thead>
<tr>
<th>The expression</th>
<th>Evaluates to</th>
<th>Interactive output</th>
</tr>
</thead>
<tbody>
<tr>
<td>delay(6)()</td>
<td>6</td>
<td>delayed 6</td>
</tr>
<tr>
<td>delay(delay)()(6)()</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>print(delay(print)()()(4))</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
def delay(arg):
    print('delayed')
    def g():
        return arg
    return g

def g():
    return arg

<table>
<thead>
<tr>
<th>The expression</th>
<th>Evaluates to</th>
<th>Interactive output</th>
</tr>
</thead>
<tbody>
<tr>
<td>delay(6)()</td>
<td>6</td>
<td>delayed 6</td>
</tr>
<tr>
<td>delay(delay)(6)()</td>
<td>6</td>
<td>delayed delayed 6</td>
</tr>
<tr>
<td>print(delay(print)(4))</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What Would Python Do? #2

```python
def delay(arg):
    print('delayed')
    def g():
        return arg
    return g
```

<table>
<thead>
<tr>
<th>The expression</th>
<th>Evaluates to</th>
<th>Interactive output</th>
</tr>
</thead>
<tbody>
<tr>
<td>delay(6)()</td>
<td>6</td>
<td>delayed 6</td>
</tr>
<tr>
<td>delay(delay)()(6)()</td>
<td>6</td>
<td>delayed delayed 6</td>
</tr>
<tr>
<td>print(delay(print)()(4))</td>
<td>None</td>
<td></td>
</tr>
</tbody>
</table>
What Would Python Do? #2

```python
def delay(arg):
    print('delayed')

def g():
    return arg
return g
```

<table>
<thead>
<tr>
<th>The expression</th>
<th>Evaluates to</th>
<th>Interactive output</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>delay(6)()</code></td>
<td>6</td>
<td><code>delayed</code></td>
</tr>
<tr>
<td><code>delay(delay)()(6)</code></td>
<td>6</td>
<td><code>delayed</code></td>
</tr>
<tr>
<td><code>print(delay(print)()(4))</code></td>
<td>None</td>
<td><code>delayed</code></td>
</tr>
</tbody>
</table>

14
What Would Python Do? #3

```python
def pirate(arggg):
    print('matey')
    def plunder(arggg):
        return arggg
    return plunder
```

<table>
<thead>
<tr>
<th>The expression</th>
<th>Evaluates to</th>
<th>Interactive output</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pirate('treasure')('scurvy')</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>add(pirate(3)(square)(4), 1)</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>pirate(pirate(pirate))(5)(7)</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.
What Would Python Do? #3

def pirate(arggg):
    print('matey')

def plunder(arggg):
    return arggg
    return plunder

<table>
<thead>
<tr>
<th>The expression</th>
<th>Evaluates to</th>
<th>Interactive output</th>
</tr>
</thead>
<tbody>
<tr>
<td>pirate('treasure')('scurvy')</td>
<td>'scurvy'</td>
<td></td>
</tr>
<tr>
<td>add(pirate(3)(square)(4), 1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pirate(pirate(pirate))(5)(7)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.
def pirate(arggg):
    print('matey')
    def plunder(arggg):
        return arggg
    return plunder

<table>
<thead>
<tr>
<th>The expression</th>
<th>Evaluates to</th>
<th>Interactive output</th>
</tr>
</thead>
<tbody>
<tr>
<td>pirate('treasure')('scurvy')</td>
<td>'scurvy'</td>
<td>matey 'scurvy'</td>
</tr>
<tr>
<td>add(pirate(3)(square)(4), 1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pirate(pirate(pirate))(5)(7)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.
```
def pirate(arggg):
    print('matey')
    def plunder(arggg):
        return arggg
    return plunder
```

<table>
<thead>
<tr>
<th>The expression</th>
<th>Evaluates to</th>
<th>Interactive output</th>
</tr>
</thead>
<tbody>
<tr>
<td>pirate('treasure')('scurvy')</td>
<td>'scurvy'</td>
<td>matey 'scurvy'</td>
</tr>
<tr>
<td>add(pirate(3)(square)(4), 1)</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>pirate(pirate(pirate))(5)(7)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.
def pirate(arggg):
    print('matey')
    def plunder(arggg):
        return arggg
    return plunder

def plunder(arggg):
    return arggg

The expression | Evaluates to | Interactive output
---|---|---
pirate('treasure')('scurvy') | 'scurvy' | matey 'scurvy'
add(pirate(3)(square)(4), 1) | 17 | matey 17
pirate(pirate(pirate))((5)(7))

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.
What Would Python Do? #3

```python
def pirate(arggg):
    print('matey')
def plunder(arggg):
    return arggg
return plunder
```

<table>
<thead>
<tr>
<th>The expression</th>
<th>Evaluates to</th>
<th>Interactive output</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pirate('treasure')('scurvy')</code></td>
<td>'scurvy'</td>
<td>matey 'scurvy'</td>
</tr>
<tr>
<td><code>add(pirate(3)(square)(4), 1)</code></td>
<td>17</td>
<td>matey 17</td>
</tr>
<tr>
<td><code>pirate(pirate(pirate))(5)(7)</code></td>
<td>Error</td>
<td></td>
</tr>
</tbody>
</table>

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.
What Would Python Do? #3

```python
def pirate(arggg):
    print('matey')

def plunder(arggg):
    return arggg

def add(pirate(3)(square)(4), 1):

return plunder
```

<table>
<thead>
<tr>
<th>The expression</th>
<th>Evaluates to</th>
<th>Interactive output</th>
</tr>
</thead>
<tbody>
<tr>
<td>pirate('treasure')('scurvy')</td>
<td>'scurvy'</td>
<td>matey 'scurvy'</td>
</tr>
<tr>
<td>add(pirate(3)(square)(4), 1)</td>
<td>17</td>
<td>matey 17</td>
</tr>
<tr>
<td>pirate(pirate(pirate))(5)(7)</td>
<td>Error</td>
<td>matey</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Error</td>
</tr>
</tbody>
</table>

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.
def horse(mask):
    horse = mask
def mask(horse):
    return horse
    return horse(horse(mask))
mask = lambda horse: horse(2)
horse(mask)

Global frame

<table>
<thead>
<tr>
<th>horse</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>mask</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

f1:

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Return value</td>
</tr>
</tbody>
</table>

f2:
f3:

Return value
def remove(n, digit):
    """Return digits of non-negative N that are not DIGIT, for some non-negative DIGIT less than 10.
    >>> remove(231, 3)
    21
    >>> remove(243132, 2)
    4313
    """
    kept = 0
digits = 0
    while ____________________________:
        last = n % 10
        n = n // 10
        if ____________________________:
            kept = ______________________
            digits = ______________________
    return ____________________________
def remove(n, digit):
    """Return digits of non-negative N that are not DIGIT, for some non-negative DIGIT less than 10.
    >>> remove(231, 3)
    21
    >>> remove(243132, 2)
    4313
    """
    kept = 0
digits = 0
    while ____________________________:
        last = n % 10
        n = n // 10
        if ____________________________:
            kept = __________
            digits = __________
    return _____________________________
Implementing a function

```python
def remove(n, digit):
    """Return digits of non-negative N that are not DIGIT, for some non-negative DIGIT less than 10.
    >>> remove(231, 3)
    21
    >>> remove(243132, 2)
    4313
    """

kept = 0
digits = 0

while __________________________:
    last = n % 10
    n = n // 10
    if __________________________:
        kept = __________________
        digits = _________________

return __________________________
```

- Read the description
- Verify the examples & pick a simple one
def remove(n, digit):
    """Return digits of non-negative N that are not DIGIT, for some non-negative DIGIT less than 10.
    >>> remove(231, 3)
    21
    >>> remove(243132, 2)
    4313
    """
    kept = 0
    digits = 0
    while ____________________________:
        last = n % 10
        n = n // 10
        if ____________________________:
            kept = ______________________
            digits = ______________________
    return ____________________________

- Read the description
- Verify the examples & pick a simple one
- Read the template
def remove(n, digit):
    """Return digits of non-negative N that are not DIGIT, for some non-negative DIGIT less than 10.
    >>> remove(231, 3)
    21
    >>> remove(243132, 2)
    4313
    ""
kept = 0
digits = 0
while ____________________________:
    last = n % 10
    n = n // 10
    if ____________________________:
        kept = __________________
        digits = _____________
return ____________________________

- Read the description
- Verify the examples & pick a simple one
- Read the template
- Implement without the template, then change your implementation to match the template.
  OR If the template is helpful, use it.
def remove(n, digit):
    """Return digits of non-negative N that are not DIGIT, for some non-negative DIGIT less than 10.
    >>> remove(231, 3)
    21
    >>> remove(243132, 2)
    4313
    """
    kept = 0
digits = 0
    while __________________________:
        last = n % 10
        n = n // 10
        if __________________________:
            kept = ____________________
            digits = ____________________
    return __________________________

- Read the description
- Verify the examples & pick a simple one
- Read the template
- Implement without the template, then change your implementation to match the template.
  OR If the template is helpful, use it.
- Annotate names with values from your chosen example
def remove(n, digit):
    '''Return digits of non-negative N that are not DIGIT, for some non-negative DIGIT less than 10.
    >>> remove(231, 3)
    21
    >>> remove(243132, 2)
    4313
    '''
    kept = 0
    digits = 0
    while ____________________________:
        last = n % 10
        n = n // 10
        if ____________________________:
            kept = __________________________
            digits = __________________________
    return ____________________________

• Read the description
• Verify the examples & pick a simple one
• Read the template
• Implement without the template, then change your implementation to match the template.
  OR If the template is helpful, use it.
• Annotate names with values from your chosen example
• Write code to compute the result
Implementing a function

```python
def remove(n, digit):
    """Return digits of non-negative N that are not DIGIT, for some non-negative DIGIT less than 10.
    >>> remove(231, 3)
    21
    >>> remove(243132, 2)
    4313
    """
    kept = 0
digits = 0
    while __________________________:
        last = n % 10
        n = n // 10
        if __________________________:
            kept = ____________________
            digits = ____________________
    return __________________________
```

- Read the description
- Verify the examples & pick a simple one
- Read the template
- Implement without the template, then change your implementation to match the template.
  OR If the template is helpful, use it.
- Annotate names with values from your chosen example
- Write code to compute the result
- Did you really return the right thing?
def remove(n, digit):
    """Return digits of non-negative N that are not DIGIT, for some non-negative DIGIT less than 10."
    >>> remove(231, 3)
    21
    >>> remove(243132, 2)
    4313
    """
    kept = 0
digits = 0
    while _____________________________:
        last = n % 10
        n = n // 10
        if _____________________________:
            kept = ______________________
            digits = ______________________
    return _____________________________

• Read the description
• Verify the examples & pick a simple one
• Read the template
• Implement without the template, then change your implementation to match the template.
  OR If the template is helpful, use it.
• Annotate names with values from your chosen example
• Write code to compute the result
• Did you really return the right thing?
• Check your solution with the other examples
Implementing a function

def remove(n, digit):
    """Return digits of non-negative N that are not DIGIT, for some non-negative DIGIT less than 10.
    >>> remove(231, 3)
    21
    >>> remove(243132, 2)
    4313
    """
    kept = 0
    digits = 0
    while n > 0:
        last = n % 10
        n = n // 10
        if last != digit:
            kept = kept + (last * 10 ** digits)
            digits = digits + 1
    return kept