Function Examples

Class outline:

- Currying
- Decorators
- Review

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:

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

```
from operator import add

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

```
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:

```
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:

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

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

Alternate approach:

```
transform_numbers(3, 4, 5, lambda x: add(60, x))
```

Use case for currying #2

Turning a generalized function into a specialized function:

```
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:

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

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

Alternate approach:

```
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.

Decorators

A tracing function

Let's make a higher-order tracing function.

```
def trace1(f):
    """Return a function that takes a single argument, x, prints it,
    computes and prints F(x), and returns the computed value.
    >>> square = lambda x: x * x
    >>> trace1(square)(3)
    -> 3
    <- 9
    9
    """
```

A tracing function

Let's make a higher-order tracing function.

```
def trace1(f):
    """Return a function that takes a single argument, x, prints it,
    computes and prints F(x), and returns the computed value.
    >>> square = lambda x: x * x
    >>> trace1(square)(3)
    -> 3
    <- 9
    9
    """
    def traced(x):
        print("->", x)
        r = f(x)
        print("<-", r)
        return r
    return traced
```

A tracing decorator

What if we always wanted a function to be traced?

```
@trace1
def square(x):
    return x * x
```

That's equivalent to..

```
def square(x):
    return x * x
square = trace1(square)
```

General decorator syntax

The notation:

```
@ATTR
def aFunc(...):
    ...
```

is essentially equivalent to:

```
def aFunc(...):
    ...
aFunc = ATTR(aFunc)
```

`ATTR` can be any expression, not just a single function name.

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.

The expression	Evaluates to	Interactive output
`5`		
`print(5)`		
`print(print(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.

The expression	Evaluates to	Interactive output
`5`	`5`	
`print(5)`		
`print(print(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.

The expression	Evaluates to	Interactive output
`5`	`5`	`5`
`print(5)`		
`print(print(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.

The expression	Evaluates to	Interactive output
`5`	`5`	`5`
`print(5)`		
`print(print(5))`		

```
>> 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.

The expression	Evaluates to	Interactive output
`5`	`5`	`5`
`print(5)`	`None`	
`print(print(5))`		

```
>> 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.

The expression	Evaluates to	Interactive output
`5`	`5`	`5`
`print(5)`	`None`	`5`
`print(print(5))`		

```
>> 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.

The expression	Evaluates to	Interactive output
`5`	`5`	`5`
`print(5)`	`None`	`5`
`print(print(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.

The expression	Evaluates to	Interactive output
`5`	`5`	`5`
`print(5)`	`None`	`5`
`print(print(5))`	`None`	

```
>> 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.

The expression	Evaluates to	Interactive output
`5`	`5`	`5`
`print(5)`	`None`	`5`
`print(print(5))`	`None`	`5` `None`

```
>> 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.

The expression	Evaluates to	Interactive output
`5`	`5`	`5`
`print(5)`	`None`	`5`
`print(print(5))`	`None`	`5` `None`

```
>> 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
```

The expression	Evaluates to	Interactive output
`delay(6)()`		
`delay(delay)()(6)()`		
`print(delay(print)()(4))`		

What Would Python Do? #2

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

The expression	Evaluates to	Interactive output
`delay(6)()`	`6`	
`delay(delay)()(6)()`		
`print(delay(print)()(4))`		

What Would Python Do? #2

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

The expression	Evaluates to	Interactive output
`delay(6)()`	`6`	`delayed` `6`
`delay(delay)()(6)()`		
`print(delay(print)()(4))`		

What Would Python Do? #2

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

The expression	Evaluates to	Interactive output
`delay(6)()`	`6`	`delayed` `6`
`delay(delay)()(6)()`	`6`	
`print(delay(print)()(4))`		

What Would Python Do? #2

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

The expression	Evaluates to	Interactive output
`delay(6)()`	`6`	`delayed` `6`

`delay(delay)()(6)()`	`6`	`delayed`
		`delayed`
		`6`

`print(delay(print)()(4))`		

What Would Python Do? #2

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

The expression	Evaluates to	Interactive output
`delay(6)()`	`6`	`delayed` `6`

`delay(delay)()(6)()`	`6`	`delayed`
		`delayed`
		`6`

`print(delay(print)()(4))`	`None`	

What Would Python Do? #2

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

The expression	Evaluates to	Interactive output
`delay(6)()`	`6`	`delayed` `6`

`delay(delay)()(6)()`	`6`	`delayed`
		`delayed`
		`6`

`print(delay(print)()(4))`	`None`	`delayed`
		`4`
		`None`

What Would Python Do? #3

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

The expression	Evaluates to	Interactive output
`pirate('treasure')('scurvy')`		
`add(pirate(3)(square)(4), 1)`		
`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

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

The expression	Evaluates to	Interactive output
`pirate('treasure')('scurvy')`	`'scurvy'`	
`add(pirate(3)(square)(4), 1)`		
`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

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

The expression	Evaluates to	Interactive output
`pirate('treasure')('scurvy')`	`'scurvy'`	`matey` `'scurvy'`
`add(pirate(3)(square)(4), 1)`		
`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

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

The expression	Evaluates to	Interactive output
`pirate('treasure')('scurvy')`	`'scurvy'`	`matey` `'scurvy'`
`add(pirate(3)(square)(4), 1)`	`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

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

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

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

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)`	`Error`	

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
```

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)`	`Error`	`matey`
		`matey`
		`Error`

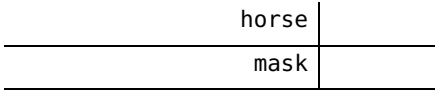
A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

Environment Diagram

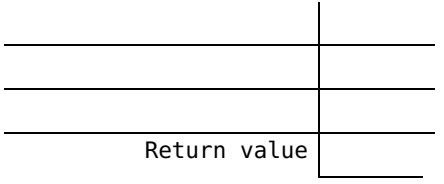
```
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)
horse(mask)
```

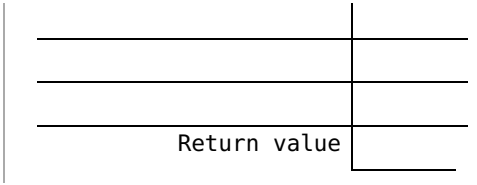
Global frame



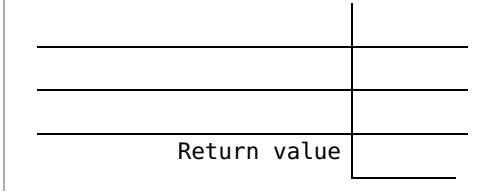
f1:



f2:



f3:



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 ________________________________:
        last = n % 10
        n = n // 10
        if ________________________________:
            kept = ____________________
            digits = ____________________
    return ____________________________________
```

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 ________________________________:
        last = n % 10
        n = n // 10
        if ________________________________:
            kept = ____________________
            digits = ____________________
    return ____________________________________
```

- Read the description

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 ________________________________:
        last = n % 10
        n = n // 10
        if ________________________________:
            kept = ____________________
            digits = ____________________
    return ____________________________________
```

- Read the description
- Verify the examples & pick a simple one

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 ________________________________:
        last = n % 10
        n = n // 10
        if ________________________________:
            kept = ____________________
            digits = ____________________
    return ____________________________________
```

- Read the description
- Verify the examples & pick a simple one
- Read the template

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 ________________________________:
        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.

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 ________________________________:
        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

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 ________________________________:
        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

```
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?

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 _____________________________:
        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
```