Midterm Review (Part 2)

Announcements

Example: Reverse

The square function can be defined in terms of the built-in pow function:

```
def square(x):
    """Square x.

    >>> square(3)
    9
    """
    return pow(x, 2)
```

```
def cube(x):
    """Cube x.

    >>> cube(3)
    27
    """
    return pow(x, 3)
```

Define square and cube in one line without using lambda or ** (using **curry** and **reverse** and **pow**).

```
def reverse(f):
    return lambda x, y: f(y, x)
```

Reverse switches the argument order of a two-argument function
reverse(pow)(2, 3) == pow(3, 2)

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

Curry converts a two-argument function into a function that takes the first and then the second argument in two calls:
curry(pow)(5)(2) == pow(5, 2)

square = _____

cube = _____

pollev.com/cs61a

Example: Reverse

The square function can be defined in terms of the built-in pow function:

```
def square(x):
    """Square x.

    >>> square(3)
    9
    """
    return pow(x, 2)
```

```
def cube(x):
    """Cube x.

    >>> cube(3)
    27
    """
    return pow(x, 3)
```

Define square and cube in one line without using lambda or ** (using **curry** and **reverse** and **pow**).

```
def reverse(f):
    return lambda x, y: f(y, x)
```

Reverse switches the argument order of a two-argument function
reverse(pow)(2, 3) == pow(3, 2)

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

Curry converts a two-argument function into a function that takes the first and then the second argument in two calls:
curry(pow)(5)(2) == pow(5, 2)

```
square = curry(reverse(pow))(2)
cube   = curry(reverse(pow))(3)
```

pollev.com/cs61a

Functional Abstraction Practice

Spring 2025 Question 3(a)

The function **all_digits** takes a positive integer **n** and one-argument function **cond** that always returns **True** or **False.** It returns **True** if **cond(d)** returns **True** when called on every digit **d** in **n,** and **False** otherwise.

```
def all_digits(n, cond):
    """Return whether cond returns true for every digit of positive n.

    >> odd = lambda d: d % 2 == 1
    >>> all_digits(123, odd)  # not all digits are odd
    False
    >>> all_digits(357, odd)  # all digits are odd
    True
    """
    while n:

        if ____________________:

            ____________________

        n = n // 10

    return __________
```

Spring 2025 Question 3(a)

The function **all_digits** takes a positive integer **n** and one-argument function **cond** that always returns **True** or **False.** It returns **True** if **cond(d)** returns **True** when called on every digit **d** in **n,** and **False** otherwise.

```
def all_digits(n, cond):
    """Return whether cond returns true for every digit of positive n.

    >> odd = lambda d: d % 2 == 1
    >>> all_digits(123, odd)  # not all digits are odd
    False
    >>> all_digits(357, odd)  # all digits are odd
    True
    """
    while n:  123
        if ____not cond(n % 10)____:
            ____return False____
        n = n // 10   12
    return ____True____
```

Do something with the 3!

pollev.com/cs61a

From discussion:

Describe a process (in English) that computes the output from the input using simple steps.

Can you ever stop before going through all of the digits?

Read the description

Verify the examples & pick a simple one

Read the template

Annotate names with values from your chosen example

Write code to compute the result

Did you really return the right thing?

Spring 2025 Question 3(a)

The function **all_digits** takes a positive integer **n** and one-argument function **cond** that always returns **True** or **False.** It returns **True** if **cond(d)** returns **True** when called on every digit **d** in **n,** and **False** otherwise.

```
def all_digits(n, cond):
    """Return whether cond returns true for every digit of positive n.

    >>> odd = lambda d: d % 2 == 1
    >>> all_digits(123, odd)  # not all digits are odd
    False
    >>> all_digits(357, odd)  # all digits are odd
    True
    """
    while n:
        if ____________________:
            ____________________
        n = n // 10
    return ____________
```

n: ~~123~~ 12

~~odd(3)~~ odd(2)

not cond(n % 10)

return False

True

From discussion:

Describe a process (in English) that computes the output from the input using simple steps.

Can you ever stop before going through all of the digits?

Read the description

Verify the examples & pick a simple one

Read the template

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

Spring 2025 Question 3(b)

From 3(a): the function **all_digits** takes a positive integer **n** and one-argument function **cond** that always returns **True** or **False**. It returns **True** if **cond(d)** returns **True** when called on every digit **d** in **n**, and **False** otherwise.

Definition. A prefix of a positive integer **n** is the value of **n//pow(10, p)** for some non-negative integer **p**. For example, 3456 has prefixes 3456, 345, 34, 3, and 0.

```
def prefix_digits(n, cond):
    """Return the largest prefix of positive n for which cond returns true for every digit.

    >>> odd = lambda d: d % 2 == 1
    >>> prefix_digits(94720, odd)
    9
    >>> prefix_digits(919321, odd)
    9193
    >>> prefix_digits(2025, odd)
    0
    >>> prefix_digits(20252025, lambda d: d < 4)
    202
    >>> prefix_digits(20252025, lambda d: True)
    20252025
    """                94720
    return process(n, lambda k: _____________________________________ )
```

```
def process(n, check):
    """A function to help implement prefix_digits."""
    while n: 94720
        if check(n): What should this do?
            return n eventually: return 9
        n = n // 10 9472
    return 0
```

Important questions:

- What does **process** do?
- How does that help with **prefix_digits**?
- What does the **k** represent in **lambda k:____**?

pollev.com/cs61a

Spring 2025 Question 3(b)

From 3(a): the function **all_digits** takes a positive integer **n** and one-argument function **cond** that always returns **True** or **False**. It returns **True** if **cond(d)** returns **True** when called on every digit **d** in **n**, and **False** otherwise.

Definition. A prefix of a positive integer **n** is the value of **n//pow(10, p)** for some non-negative integer **p**. For example, 3456 has prefixes 3456, 345, 34, 3, and 0.

```
def prefix_digits(n, cond):
    """Return the largest prefix of positive n for which cond returns true for every digit.

    >>> odd = lambda d: d % 2 == 1
    >>> prefix_digits(94720, odd)
    9
    >>> prefix_digits(919321, odd)
    9193
    >>> prefix_digits(2025, odd)
    0
    >>> prefix_digits(20252025, lambda d: d < 4)
    202
    >>> prefix_digits(20252025, lambda d: True)
    20252025
    """
    return process(n, lambda k: _____)
```

94720

all_digits(k, cond)

```
def process(n, check):
    """A function to help implement prefix_digits."""
    while n: 94720
        if check(n): What should this do?
            return n eventually: return 9
        n = n // 10 9472
    return 0
```

Important questions:

- What does **process** do?
- How does that help with **prefix_digits**?
- What does the **k** represent in **lambda k:____**?

pollev.com/cs61a

Iteration Practice

Spring 2025 Question 3(c)

Definition. A prefix of a positive integer **n** is the value of **n//pow(10, p)** for some non-negative integer **p.** For example, 3456 has prefixes 3456, 345, 34, 3, and 0.

```
def prefix_digits(n, cond):
    """Return the largest prefix of positive n for which cond returns true for every digit.
    >>> odd = lambda d: d % 2 == 1
    >>> prefix_digits(94720, odd)
    9
    >>> prefix_digits(919321, odd)
    9193
    >>> prefix_digits(2025, odd)
    0
    >>> prefix_digits(20252025, lambda d: d < 4)
    202
    """
    k = 0
    while n >= ____________________________ :
        if cond(____________________________):
            k += 1
        else:
            n = n // 10
            ____________________________
    return n
```

Questions to help figure out the process:

- Can you ever stop before going through all of the digits?
- What kind of value should you return?
- What does k represent?
- How does increasing k help?

Spring 2025 Question 3(c)

Definition. A prefix of a positive integer **n** is the value of **n//pow(10, p)** for some non-negative integer **p.** For example, 3456 has prefixes 3456, 345, 34, 3, and 0.

```
def prefix_digits(n, cond):
    """Return the largest prefix of positive n for which cond returns true for every digit.
    >>> odd = lambda d: d % 2 == 1
    >>> prefix_digits(94720, odd)
    9
    >>> prefix_digits(919321, odd)
    9193
    >>> prefix_digits(2025, odd)
    0
    >>> prefix_digits(20252025, lambda d: d < 4)
    202
    """
    k = 0  1  0  1
    while n >= ____pow(10, k)____ :
        if cond(__n // pow(10, k) % 10__):
            k += 1
        else:  n: 94720  9472  947  94  9
            n = n // 10
            k = 0
    return n  eventually: 9
```

pollev.com/cs61a

pollev.com/cs61a

Questions to help figure out the process:

- Can you ever stop before going through all of the digits?
- What kind of value should you return?
- What does k represent?
- How does increasing k help?

Process that fits this template:

- Use k to call cond on each digit of n
- If cond ever returns false, shorten n and start over

```
odd(0) -> False
odd(2) -> False
odd(7) -> True
odd(4) -> False
odd(4) -> False
odd(9) -> True
```

Call cond on the last digit

Call cond on the second to last digit

Spring 2025 Question 4

```
def hailstone(n):
    """Print numbered updates in the hailstone sequence.

    >>> hailstone(10)
    1 10 -> 5
    2 5 -> 16
    3 16 -> 8
    4 8 -> 4
    5 4 -> 2
    6 2 -> 1
    """
    def f():
        if n % 2 == 1:
            m = 3 * n + 1
        else:
            m = n // 2

        ____________________________________

        ____________________________________

    k = 1
    while n > 1:
        k, n = k + 1, f()
```

Questions to help figure out the process:

- What does **f** do?
- What do **m**, **n**, and **k** represent?

Spring 2025 Question 4

```
def hailstone(n):
    """Print numbered updates in the hailstone sequence.

    >>> hailstone(10)
    1 10 -> 5
    2 5 -> 16
    3 16 -> 8
    4 8 -> 4
    5 4 -> 2
    6 2 -> 1
    """
    def f():
        if n % 2 == 1:
            m = 3 * n + 1
        else:
            m = n // 2
        print(k, n, '->', m)
        return m
    k = 1
    while n > 1:
        k, n = k + 1, f()
```

m: 5

k: 1

n: 10

Questions to help figure out the process:

- What does **f** do?
- What do **m**, **n**, and **k** represent?

Lambda Practice

Lambda and Def

Any program containing lambda expressions can be rewritten using def statements.

```
                          twice                          square
>>> (lambda f: lambda x: f(f(x)))(lambda y: y * y)(3)
81

>>> def twice(f):
...     def g(x):
...         return f(f(x))
...     return g
...
>>> def square(y):
...     return y * y
...
>>> twice(square)(3)
81
```