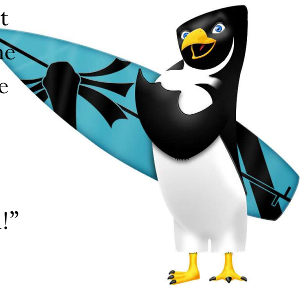
THE CS 5 BLACK TIMES

Penguin Leads Surfing Trip

Oceanside (UPI)—In an attempt to make up for the chaos caused by her ill-behaved colleagues during a Harvey Mudd College, a well-tanned penguin took a group of first-year students on a beach outing in which they learned to surf. "It was *awesome,*" gushed one surfer. I stood up the very first time and I would have made it all the way in except that I slipped on some fish oil."

Another beginner lauded the experience even though he returned with gashes on his chin. "I got to see the ocean from underneath!" he exclaimed. "I had no idea there were fish under there!"

Read Sections 2.10-2.12



Getting Help and Office Hours

Come to office hours or set up times to come talk to Geoff and Zach! Also, grutoring hours are great!

Be sure to put "CS5" in your e-mail subject lines!

Check your e-mail for Zoom links!

Filter

Java doesn't have a `filter`!

```
def even(x):
   '''Returns True iff x is even'''
   return x % 2 == 0
```

A function that returns either `True` or `False` Is called a *predicate*

```
>>> list(filter(even, range(100)))
[0, 2, 4, 6, …, 98]
```

Filter

```
def short(List):
   '''Returns True iff List has len <= 2'''
   return len(List) <= 2

>>> list(filter(short, [["spam", "yum"], [42], [1, 2, 3]]))
```

`filter` can be written from scratch using recursion.

Functions are Data

```
def divides(n):
  def div(k):
      return n % k == 0
  return div

>>> div10 = divides(10)
>>> div10
<function div10 at 0x661f0>
>>> div10(2)

>>> listOfFunctions = [divides(10), divides(20)]
>>> listOfFunctions[0](2)
```

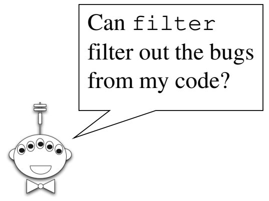
Anonymous Functions

```
filter(   e  n(x): return x%2 == 0,
  range   00

filter(    (   : return x%2 == 0,
  range   00

filter(lambda x: x%2 == 0, range(100))
```

One line
no parentheses on the argument
`return` is implicit

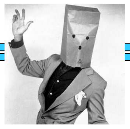
Alonzo Church
1903-1955

```
>>> lambda_dbl = lambda x: 2 * x
>>> lambda_dbl(21)
42
```



Lambda

aka “anonymous functions”



```
>>> list(filter(lambda x: x%2 == 0,
        range(100)))

>>> list(filter(lambda List: len(List) <= 2,
        [["spam", "yum"], [42], [1, 2, 3]]))
```

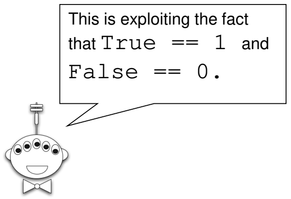
Lambda

```
even = lambda x: x%2 == 0

def even(x):
   '''Returns True iff x is even'''
   return x % 2 == 0
```

```
short = lambda List: len(List) <= 2

def short(List):
   '''Returns True iff List has len <= 2'''
   return len(List) <= 2
```

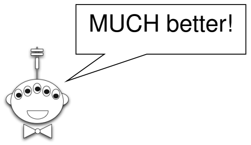
Lambda Evil

```
def ugly(item, L):
  newL = list(map(lambda x: x == item, L))
  return sum(newL) > 0
```



Lambda

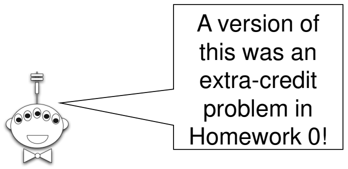
```
from functools import reduce

def mystery(item, L):
  newL = list(map(lambda x: x == item, L))
  return reduce(lambda x, y: x or y, newL)
```



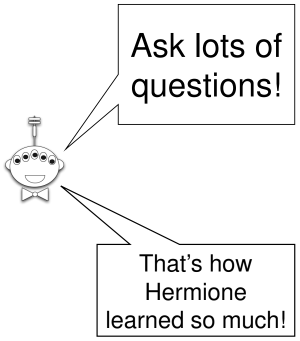
A Prime Example

Write a function called `prime(n)` that returns `True` if `n` is prime and `False` otherwise by testing all possible divisors from `2` to `n-1` (or sqrt of `n`)

```
def prime(n):
   possibleDivisors = range(2, n)
   divisors = filter(                                   )
   return ???
```


A version of this was an extra-credit problem in Homework 0!

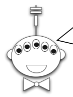
The Alien's Life Advice



Use-It-Or-Lose-It

Power Set!

```
>>> powerset([1, 2])
[[], [2], [1], [1, 2]]

>>> powerset([1, 2, 3])
[[], [3], [2], [2, 3], [1], [1, 3],
  [1, 2], [1, 2, 3]]
>>> powerset([1])


>>> powerset([])
```



This really demonstrates the power of functional programming!

The order in which the subsets are presented is unimportant but within each subset, the order should be consistent with the input set.

Power Set!

```
def powerset(L):
```

In your notes…

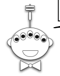
The Knapsack Problem…

Kingdom of Shmorbodia

Item	Weight	Value
Spam	2	100
Tofu	3	112
Chocolate	4	125

Knapsack Capacity: 5? 6? 7?

```
>>> knapsack(7, [[2, 100], [3, 112], [4, 125]])
237
```



Prof. I. Lai thinks that a "greedy solution" is the way to go!

Worksheet and Demo

The Knapsack Revisited…

Kingdom of Shmorbodia

Item	Weight	Value
Spam	2	100
Tofu	3	112
Chocolate	4	125

Knapsack Capacity: 5? 6? 7?

```
>>> knapsack(7, [[2, 100], [3, 112], [4, 125]])
[237, [[3, 112], [4, 125]]]
```

Comparing DNA via Longest Common Subsequence (LCS)

AGGACAT
ATTACGAT

```
>>> LCS("AGGACAT", "ATTACGAT")
5
>>> LCS("spam", "sam!")
3
>>> LCS("spam", "xsam")
3
```

I prefer spam to an xsam!

Recursive Approach…

```
def LCS(S1, S2):
    if BASE CASE
    else:
```

LCS("spam", "sam!")

Try this in your notes!
Solution follows