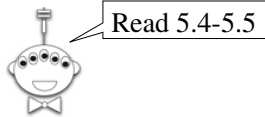


The CS 5 Black Gazette

CS 5 Penguins Located at Massive Penguin Party

(AP) The two missing CS 5 “Black” penguins surfaced on a large ice floe near Antarctica. Authorities indicated that an enormous penguin dance party was underway there that would likely last well into Novembrrrr. The CS 5 instructors were relieved to hear that the penguins are safe. “I’m glad they’re OK, but I’m a bit disappointed that I wasn’t invited to this very ‘cool’ party,” said one of the profs.



Join the ACM for Free!



- World’s most important society for computer scientists
- Publishes cutting-edge research
- Many, many benefits

Just visit <https://www.acm.org/studentjoin>

It’s easy...and free!

About the Midterm

- Thursday, October 22nd
- Comprehensive through 10/15
- You should definitely know:
 - Recursion (including multiple base cases)
 - map, reduce, filter, lambda
 - List comprehension: $[x*2 \text{ for } x \text{ in } L]$
 - Higher-order functions (functions that return functions)
 - Use-it-or-lose-it

More You Should Know

- Logic circuits
 - AND, OR, NOT
 - Writing truth tables
 - **Minterm expansion principle**
 - Using AND and OR to choose an output
- Hmmm programming
 - Recursion techniques
 - Conditional jumps
 - (We will supply a Hmmm reference sheet)

Yet More

- Simple imperative programming
 - Assignment statements
 - If/else/elif
 - For loops (for i in *iterable*)
 - While loops
 - Break and continue
 - Return from inside loops
 - Try/except

Gradescope Setup

- Exam will be released on 10/21 at 8 AM
- Due time will be 10/22 at 1:00 PM
- If you want during-exam support, take it in class on Thursday morning (I will be there to answer questions in the chat)
 - Otherwise feel free to sleep in!
- Closed book, under HMC Honor Code (see next slide)

Your Cheat Sheet

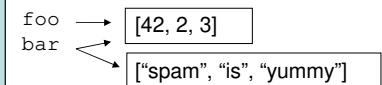
You're allowed ONE sheet of 8.5x11 paper, with contents up to you



Double-sided!

Deep vs. Shallow Copy

```
>>> foo = [1, 2, 3]
>>> bar = foo
>>> bar[0] = 42
>>> foo
[42, 2, 3]
```



```
>>> bar = ["spam", "is", "yummy"]
>>> foo
[42, 2, 3]
```

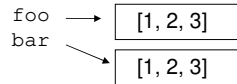
```
>>> x = 42
>>> y = x
>>> y = 57
>>> x
42
```



This is a deep concept!

Deep vs. Shallow Copy

```
>>> from copy import *
>>> foo = [1, 2, 3]
>>> bar = deepcopy(foo)
>>> bar[0] = 42
>>> foo
[1, 2, 3]
```



This is a deep concept!

Isn't Mutability Always Preferable?

```
def ben(List):
    if List != []:
        List[0] = -1

def jerry():
    myList = [1, 2, 3]
    ben(myList)
    print("My list is", myList)
```

```
>>> jerry()
My list is [-1, 2, 3]
```



Watch out here!

Isn't Mutability Always Preferable?

```
def ben(List):
    List = ["yowza!"]
```

```
def jerry():
    myList = [1, 2, 3]
    ben(myList)
    print("My list is", myList)
```

```
>>> jerry()
My list is [1, 2, 3]
```



Watch out here!

Isn't Mutability Always Preferable?

```
def ben(x):
    x = 43
```

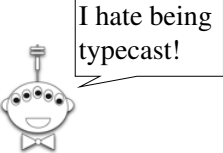
```
def jerry():
    myNum = 42
    ben(myNum)
    print("My number is", myNum)
```

```
>>> jerry()
My number is 42
```



Type Casting

```
>>> int(1.3)
1
>>> str(1.3)
'1.3'
>>> str(2+3j)
'(2+3j)'
>>> str([1, 2, 3])
'[1, 2, 3]'
>>> str("hello")
'hello'
>>> int("hello")
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ValueError: invalid literal for int(): hello
```



Non-Built-In "Types"

```
>>> from decimal import *
>>> getcontext().prec = 20
>>> x = Decimal(1)/Decimal(3)
>>> x
Decimal("0.33333333333333333333")
>>> x + 1
Decimal("1.33333333333333333333")
>>> int(x)
0
>>> float(x)
0.3333333333333333
>>> str(x)
'0.33333333333333333333'
```

Notice that argument to Decimal can be a string, an integer, or (inaccurately) a floating point number!

```
>>> Decimal(0.333)
0.333000000000000000
182076576038525672
629475593566894531
25
```

Input and Output

```
def get_input():
    """Takes no arguments. Queries user for name
    and age and returns a list [name, age] where
    name is a string and age is an integer."""
    name = input("Enter your name: ")
    age = int(input("Enter your age: "))
    return [name, age]
```

Worksheet: write a function that asks the user for a number n , asks the user to enter n integers, and returns the list of those n integers. Be robust against input errors. Helper functions encouraged!



Funky Series!

- Harmonic Sequence:
 $1/1 + 1/2 + 1/3 + 1/4 + 1/5 + 1/6 + \dots$
- Without composites (primes only):
 $1/1 + 1/2 + 1/3 + \cancel{1/4} + 1/5 + \cancel{1/6} + 1/7 + \dots$
- Without 9's...
 $1/1 + 1/2 + \dots + 1/8 + \cancel{1/9} + \dots + 1/18 + \cancel{1/20} + \dots + 1/88 + \cancel{1/89} + \cancel{1/90} + \cancel{1/91} + \dots$

Look-And-Say Sequences (aka “Read-It-And-Weep”)

1
11
21
1211
111221
312211
?



I wonder how many digits there are in the n^{th} term of this sequence?

Number of digits in the n^{th} term of the sequence is asymptotically

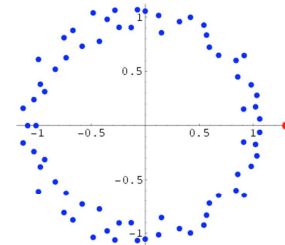
$$C \lambda^n$$

$$C = 1.567\dots$$

$$\lambda = 1.30357726034296\dots$$

Conway's Constant

Look-And-Say Sequences (aka “Read-It-And-Weep”)



The quantity λ is known as *Conway's constant* (Sloane's A014715), and amazingly is given by the unique positive real root of the polynomial

$$0 = x^{71} - x^{69} - 2x^{68} - x^{67} + 2x^{66} + x^{65} + x^{64} - x^{63} - x^{62} - x^{61} - x^{60} - x^{59} + 2x^{58} + 5x^{57} + 3x^{56} - 2x^{55} - 10x^{54} - 3x^{53} - 2x^{52} + 6x^{51} + 6x^{50} + x^{49} + 9x^{48} - 3x^{47} - 7x^{46} - 8x^{45} - 8x^{44} + 10x^{43} + 6x^{42} + 8x^{41} - 5x^{40} - 12x^{39} + 7x^{38} - 7x^{37} + 7x^{36} + x^{35} - 3x^{34} + 10x^{33} + x^{32} - 6x^{31} - 2x^{30} - 10x^{29} - 3x^{28} + 2x^{27} + 9x^{26} - 3x^{25} + 14x^{24} - 8x^{23} - 7x^{22} + 9x^{21} - 3x^{20} - 4x^{19} - 10x^{18} - 7x^{17} + 12x^{16} + 7x^{15} + 2x^{14} + 2x^{13} - 12x^{12} - 4x^{11} - 2x^{10} - 5x^9 + x^7 - 7x^6 + 7x^5 - 4x^4 + 12x^3 - 6x^2 + 3x - 6;$$

Number of digits in the n^{th} term of the sequence is asymptotically

$$C \lambda^n$$

$$C = 1.567\dots$$

$$\lambda = 1.30357726034296\dots$$

Conway's Constant

Nim and the Nim Sum



I don't know about nim sum, but I sure do love dim sum!



1. Start with n heaps of stones
2. Each player can remove any nonzero number of stones from *one* heap
3. The player who removes the last stone wins

Nim Summary



Oooh, that's **BAD**!



- Calculate nim sum as **xor** of heap sizes
- Force nim sum to be zero
 - Pick a heap where highest *nonzero* nim-sum bit matches corresponding heap bit
 - Final size should be heap size **xor** nim sum
- See Wikipedia for how to win “loser takes last” game