

String Formats

CS 8: Introduction to Computer Science, Winter 2019
Lecture #10

Ziad Matni, Ph.D.
Dept. of Computer Science, UCSB

Administrative

- **Please note that next Monday (2/18) is a Uni. Holiday**
- **Hw05 – due next week Wednesday (2/20)**
- **Lab05 is due Monday 2/18**
- Midterm Exam #1 grades will be posted today

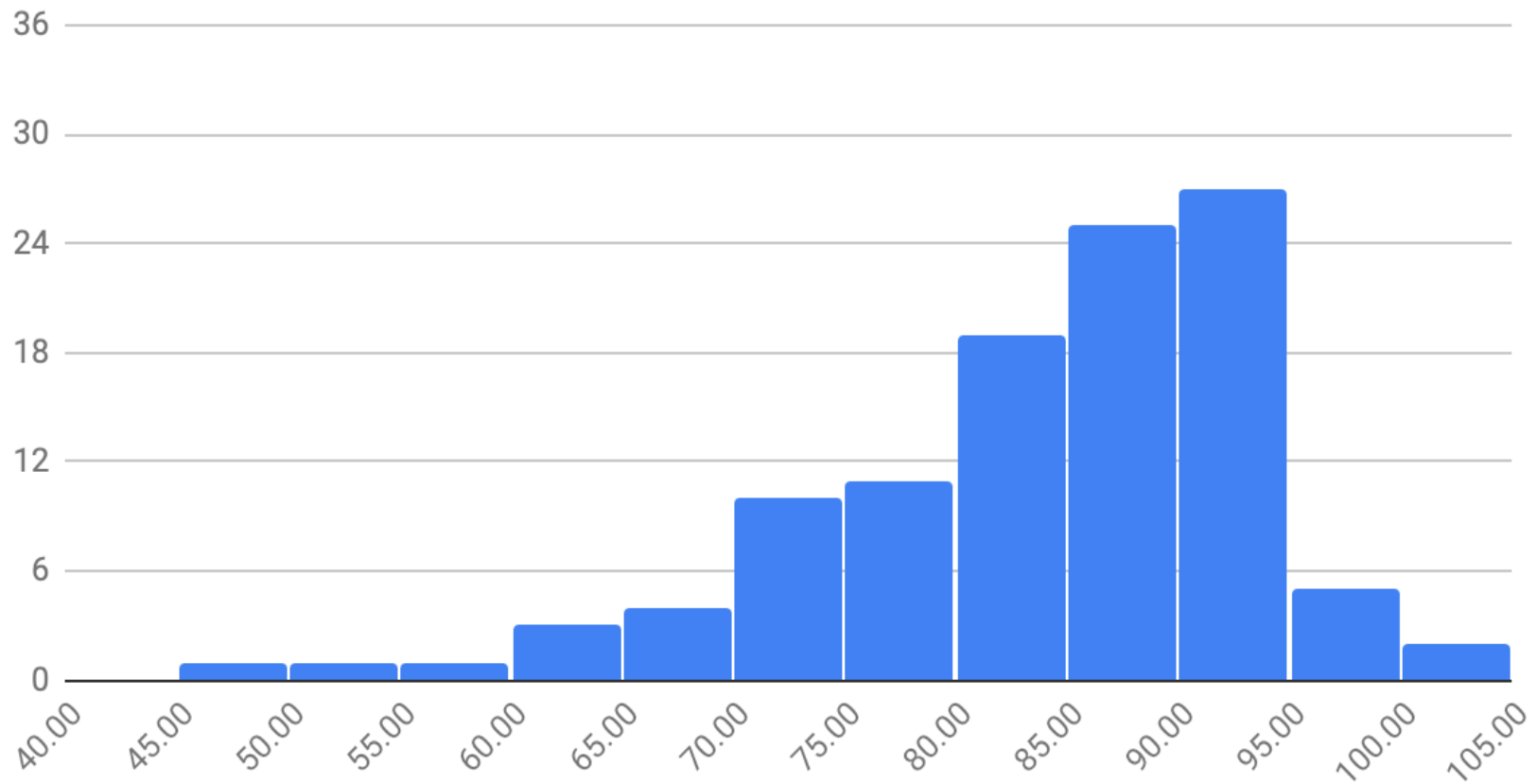
Participation Target
HIT!!!

Everyone got +2 points
on Midterm Score!



CS 8, W 19 Midterm Exam #1 Distribution

Av. = 83.1 Median = 85



Reviewing Your Midterm #1 Exam

- Go to your lab TA's **office hours** (listed in the syllabus) to review your exam
 - Exception: **students in the 1 PM lab**, see **Prof. Matni**
- When reviewing your exams:
 - Do not take pictures, do not copy the questions
 - TA cannot change your grade
 - If you have a legitimate case for grade change, the prof. will decide
 - Legitimate case = When we graded, we added the total points wrong
 - Not legitimate case = Why did you take off N points on this question????

Your Awesome Feedback!

- Most of you said
 - Prof. is very clear in lecture (yay!)
 - Goes at an appropriate pace (good!)
 - Assignments are medium-to-challenging (perfect!)
 - You are enjoying the class (woooo!)
- Some of you said
 - Too many examples / Not enough examples
 - Lab time is short
 - Please don't "cold call" on students!
 - Use the chalkboard more!

Participation Target
HIT!!!

Everyone got +2 points
on Midterm Score!



Lecture Outline

- Accumulated Loops
- String Formats

Exercises with Accumulation 3

- Useful for "accumulating" something while going through a collection.

- Finish this function:

```
def countWords(sentence):
```

```
    """ returns the number of words in the string sentence """
```

Exercises with Accumulation 3

- Useful for "accumulating" something while going through a collection.

- Finish this function:

```
def countWords(sentence):  
    """ returns the number of words in the string sentence """  
    sum = 0  
    for count in range(len(sentence)):  
        if (sentence[count] == " ") or (count == len(sentence) - 1):  
            sum += 1  
    return sum
```

The `.append` Function for Lists

- You can add items into a list by *appending* them to the end of the list
- Example: To grow `l = [1, 2]` into `l = [1, 2, 3]` you can do:
`l.append(3)`
- It's not the only way to “grow” a list, but it's easy and intuitive...

Exercises with Accumulation 4

- Useful for "accumulating" something while going through a collection.

- Finish this function:

```
def createListOfOdd(lst):  
    """ returns a new list that contains all """  
    """ the odd numbers in lst """
```

Exercises with Accumulation 4

- Useful for "accumulating" something while going through a collection.

- Finish this function:

```
def createListOfOdd(lst):  
    """ returns a new list that contains all """  
    """ the odd numbers in lst """  
    newList = []  
    for item in lst:  
        if item % 2 != 0:  
            newList.append(item)  
    return newList
```


String Delimiters

- Recall that:
“hello” and ‘hello’
are the same thing
(Python lets you use either single or double quote marks for string delimiters)
- They can even be used together, like this:
s = “hello, I’m Joe” or
s = ‘So I said, “Who are you?”’
- Otherwise, we’d have to use the \ (called “escape sequence”), like this:
s = “So I said, \”Who are you?\””

Newlines in Python

- The most straight-forward way is to use the “`\n`” character
- Example:

```
>>> s = "How I wish you were here.\nWe're just two lost souls  
swimming in a fishbowl,\nYear after year"
```

```
>>> print(s)
```

How I wish you were here.

We're just two lost souls swimming in a fishbowl,

Year after year

Note: there's no need for a third `\n` here, because the `print()` function always puts one there, BY DEFAULT (can be over-ridden)

Alternative Way to Make Newlines

- You can define a string with triple double-quotes (""""), like this:

```
>>> s = """  
How I wish you were here.  
We're just two lost souls swimming in a fishbowl,  
Year after year  
"""
```

```
>>> print(s)  
How I wish you were here.  
We're just two lost souls swimming in a fishbowl,  
Year after year
```

Recall: String Indexing & Slicing

- If `s = "hello"`
- Then `s[0] = "h"` , etc...
- The last character in any string is...
`s[len(s) - 1]`
- In the example above, `s[0:3] = "hel"`
 - In other words, it goes from index 0 to index 2 (*one-before-3*)
- Also, `s[2:] = "llo"` (from 2 to the end)
- And, `s[:4] = "hell"` (from the beg. to 3)

Negative Indices in Strings

- If `s = "hello"`
- Then `s[-1] = "o"`
`s[-2] = "l"` , etc...
- In the example above, `s[-2:] = "lo"`
etc...

Slicing Works on Lists Too!

Example:

```
ThisList = [3, 4, "spaghetti", -5]
```

```
ThisList[0:2] = [3, 4]
```

```
ThisList[-2:] = ["spaghetti", -5]
```

Recall: String Methods

SEE **TABLE 4.1**
in textbook

Assume: name = 'Bubba'

- `name.center(9)` is `' Bubba '` ← centers w/ spaces on each side
- `name.count('b')` is `2` ← counts how many times 'b' occurs
- `name.count('ubb')` is `1` ← counts how many times 'ubb' occurs
- `name.ljust(9)` is `'Bubba '` ← left justifies name in 9 spaces
- `name.rjust(9)` is `' Bubba'` ← right justifies name in 9 spaces
- `name.upper()` is `'BUBBA'` ← all uppercase letters
- `name.lower()` is `'bubba'` ← all lowercase letters
- `name.index('bb')` is `2` ← Index of first occurrence of first letter
- `name.find('bb')` is `2` ← Index of first occurrence of first letter
- `name.find('z')` is `-1` if not found, then returns -1
- `name.replace('bb','dd')` is `'Budda'` ← Replaces one sub-string for another

The `.split()` Method for Strings

- You can **split** a string into its component words and then **place them in a list**
 - With ONE instruction!!

Example:

```
>>> s = "What about Bob?"  
>>> l = s.split()  
>>> print(l)  
["What", "about", "Bob?"]
```

Note: the split is done on SPACE characters and these are NOT part of the collected sub-strings in the list!

The `.split()` Method for Strings

- The default split is on space characters (" ")
- You can over-ride that default and split on ANY string

Example:

```
>>> s = "What about Bob?"
```

```
>>> l = s.split('a')
```

```
>>> print(l)
```

```
["Wh", "t ", "bout Bob?"]
```

Note: NOW the split is done on the 'a' characters and these are NOT part of the collected sub-strings in the list!

LET'S REDO THIS EXERCISE!!!

- Finish this function:

```
def countWords(sentence):
```

```
    """ returns the number of words in the string sentence """
```

```
    sum = 0
```

```
    MyNiceList = sentence.split()
```

```
    return len(MyNiceList)
```

```
# SOOOO much easier!!!
```

Formatted Outputs

- You know these already:

```
print(42)          # prints 42 and then a newline (wow)
print(42, "!")      # prints '42 !' and then a newline (note the space)
print(42, end="")   # prints 42 WITHOUT a newline character
```

- Expanding on the above...

```
print(42, end="!")  # prints 42! WITHOUT a newline character (note NO space!)
```

Using the .format() Function with Strings

- You can print an output while you **define** your general format!

Example:

```
hour = 12  
minute = 55  
second = 31
```

*Note: the {0} refers to hour (the 0th argument),
the {1} to minute (the 1st argument), etc...*

THIS ORDER MATTERS!!

*Example, what would happen
if I switched {0} and {1} in here?*

If you do this: `'{0}:{1}:{2}'.format(hour, minute, second)`

You get this: **12:55:31** *(it's a string output)*

More on .format()

- You can define how many spaces an object occupies when printed

Example:

```
>>> a = 19
```

```
>>> b = 42
```

```
>>> '{0:3}***{1:5}'.format(a, b)
```

' 19*** 42'

3 spaces 5 spaces

*Refers to the 0th item (that is, variable **a**)*

Refers to the total number of spaces you want to format

Let's try it out!

YET MORE on .format()

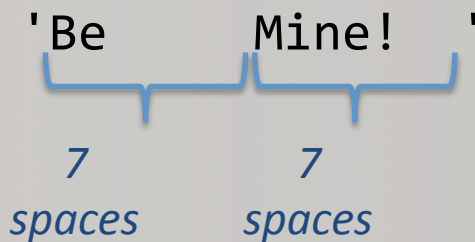
- With strings instead of numbers

Example:

```
>>> a = "Be"
```

```
>>> b = "Mine!"
```

```
>>> '{:7}{:7}'.format(a, b)
```



*Save 7 spaces for each of var. **a** and **b**
Put any extra spaces AFTER them*

What happens if you run out of space?

Does it:

***a.** cut out the string to make it fit?*

***b.** still print out the string even if
it's longer than the space format?*

Let's try it out!

.format() with Floating Points

- If you say, `print(100/3)`, you get: 33.333333333333336
- What if you wanted to instill some precision on your decimal values?

Example:

```
>>> '{:7.2}'.format(100/3)
```

' 33.33'

7

spaces

*Save 7 spaces for the floating point.
Put 2 numbers after the decimal point.
NOTE: the decimal point does take up a space*

Let's try it out!

.format() with Floating Points using Engineering Notation

- If you say, `print(100/3)`, you get: 33.333333333333336

Example:

```
>>> '{:10.1e}'.format(100/3)
```

' 3.3e+01'

10
spaces

Save 10 spaces for the floating point and use engineering notation.

Let's try it out!

More Examples

- Go to your textbook and read through all the examples in **Ch. 4.2**
- There are other types of format
- **CHECK THOSE OUT TOO!!!**

YOUR TO-DOs

- ❑ **Lab5** (turn it in by **Monday, 2/18**)
- ❑ **HW5** (due on **Wednesday, 2/20**)
- ❑ Don't eat too much candy for Valentine's
 - ❑ *Save that vice for Halloween...!*

</LECTURE>