IMPERIAL
BUSINESS SCHOOL

Data Structures and Algorithms

Lecture 3

Ryan Cory-Wright
r.cory-wright@imperial.ac.uk

Warm-Up: A Brain Teaser

This question used to be asked in FAANG interviews:

> *You are trying to cook an egg for exactly 15 minutes. However, instead of a timer, you are given two ropes which burn for exactly 1 hour each. The ropes, however, are of uneven densities, i.e., half the rope lengthwise might take only two minutes to burn.*

Warm-Up: A Brain Teaser

This question used to be asked in FAANG interviews:

> *You are trying to cook an egg for exactly 15 minutes. However, instead of a timer, you are given two ropes which burn for exactly 1 hour each. The ropes, however, are of uneven densities, i.e., half the rope lengthwise might take only two minutes to burn.*

What do interviewers look for here? A systematic and logical approach.

- What is important here? The numbers usually have a meaning. 15 minutes, two ropes, 60 minutes are all related.

Warm-Up: A Brain Teaser

This question used to be asked in FAANG interviews:

> *You are trying to cook an egg for exactly 15 minutes. However, instead of a timer, you are given two ropes which burn for exactly 1 hour each. The ropes, however, are of uneven densities, i.e., half the rope lengthwise might take only two minutes to burn.*

What do interviewers look for here? A systematic and logical approach.

- What is important here? The numbers usually have a meaning. 15 minutes, two ropes, 60 minutes are all related.
- What can we do immediately? Measure one hour by burning a rope.

Warm-Up: A Brain Teaser

This question used to be asked in FAANG interviews:

> *You are trying to cook an egg for exactly 15 minutes. However, instead of a timer, you are given two ropes which burn for exactly 1 hour each. The ropes, however, are of uneven densities, i.e., half the rope lengthwise might take only two minutes to burn.*

What do interviewers look for here? A systematic and logical approach.

- What is important here? The numbers usually have a meaning. 15 minutes, two ropes, 60 minutes are all related.
- What can we do immediately? Measure one hour by burning a rope.
- Insight 1: we can also measure 30 mins, by burning a rope at both ends

Warm-Up: A Brain Teaser

This question used to be asked in FAANG interviews:

> *You are trying to cook an egg for exactly 15 minutes. However, instead of a timer, you are given two ropes which burn for exactly 1 hour each. The ropes, however, are of uneven densities, i.e., half the rope lengthwise might take only two minutes to burn.*

What do interviewers look for here? A systematic and logical approach.

- What is important here? The numbers usually have a meaning. 15 minutes, two ropes, 60 minutes are all related.
- What can we do immediately? Measure one hour by burning a rope.
- Insight 1: we can also measure 30 mins, by burning a rope at both ends
- Solution: we can also measure 15 mins, by lighting one rope at both ends and the other rope at one end. When the first rope is done, light the other end of the second one and start cooking your egg. When the second rope is done, your egg is cooked.

Today

1. **Recap**
2. Lists and `for` loops
   - Strings
   - Review of prepared material
3. Problem-solving and debugging
   - Some more challenging exercises

Intended Learning Outcomes

- Write for loops over range() and over sequences (lists/strings), including nested loops
- Create and manipulate lists
- Turn a problem statement into a clear algorithmic plan, then implement and test it on examples
- Describe the core idea of binary search and why it needs about $\log_2 n$ guesses

So far

- Variables allow us to store results
- Conditional statements allow us to make decisions
- While loops let us repeat things
- Algorithms are step-by-step instructions
- Functions let us give names to procedures

What is the output sequence?

```python
i = 4
while i > 2:
    print(i*i)
i = i - 1
```

A. 16, 9, 4

B. 16, 9

C. 9, 4

D. An infinite loop

E. I don’t know

What is the output?

```python
def fun_function(a, b):
    if b > a:
        return b
    print(a)
x = fun_function(3, 4)
print(x)
```

A. 4

B. 3

C. 3, then 4

D. 4, then 3

E. I don’t know

What is the output?

```python
def fun_function(a, b):
    if b > a:
        print(2*b)
    print(a)
x = fun_function(-3, -1)
print(abs(x))
```

A. -2

B. -2, -3, 3

C. -2, -3, Error

D. -2, Error

E. I don’t know

Recap: Some Python Data Types

Name	Python Declaration	Example
Boolean	`bool`	`x = True`
Integer	`int`	`x = 5`
Floating Point Number	`float`	`x = 5.0`
String	`str`	`x = 'Hello world'`
List	`list`	`x = ['Hello', 'There']`
		`y = [2, 'Hi']`

Variables store data of different types, and these different types have different memory costs and can do different things.

Today

1. Recap
2. **Lists and `for` loops**
3. Problem solving and debugging

For loops

We know how to use while loops:

```python
i = 0
while i < 4:
    print(i)
    i = i + 1
```

We can also use a for loop:

```python
# range(n) generates integers 0, ..., n-1
for i in range(4):
    print(i)
```

General form:

```python
for item in sequence:
    # do something with item
```

Range:

```python
# range(start, end, step)
# step can be negative, for example range(10, 7, -1)
for i in range(5, 11, 2):
    print(i)
```

What is the output sequence?

```python
for i in range(1, 4):
    print(i)
    print(i*i)
```

A. 2, 4, 3, 9, 4, 16

B. 1, 1, 2, 4, 3, 9, 4, 16

C. 1, 1, 2, 4, 3, 9

D. 2, 4, 3, 9

E. I don’t know

How many times is ’Hello’ printed?

```python
for i in range(1, 4):
    print(i)
print('Hello')
```

A. 0

B. 1

C. 3

D. 4

E. I don’t know

What is the output?

```python
s = 0
for i in range(3):
    for j in range(2):
        s += 1
print(s)
```

A. 0

B. 2

C. 3

D. 6

E. I don’t know

Today

1. Recap
2. **Sequential data and `for` loops**
    - Lists
    - Strings
3. Problem solving and debugging

Which of these reverses the string?

A:

```python
s = 'hiphop'
new_s = s[0:len(s):1]
```

B:

```python
s = 'hiphop'
new_s = s[::-1]
```

C:

```python
s = 'hiphop'
new_s = ''
for char in s:
    new_s = new_s + char
```

D:

```python
s = 'hiphop'
new_s = ''
for i in range(len(s)):
    new_s += s[len(s) - i - 1]
```

E: More than one of the above

Lists

So far we have worked with simple data like numbers.

Often we have (ordered) compound data, for example stock prices.

```python
prices = [5.2, 4.9, 5.1, 5.6, 7.0]
```

List syntax: square brackets, comma-separated values (which can be a mix of any Python objects).

E.g., a vector (1d list) or a matrix (2d list)

List example

```
>>> names = [] # create empty list
>>> names.append('Ruth') # add an item to the end of list
>>> names.append('Mo')
>>> print(names)
['Ruth', 'Mo']
>>> print(len(names))
2
>>> print(names[0]) # indexing: 0, 1, 2, ..., we can also "slice": names[1:2]
Ruth
>>> names[1] = 'Olivia' # change Mo -> Olivia
>>> for name in names: # convenient for a for loop!
>>>     print('Hello ' + name)
Hello Ruth
Hello Olivia
>>> # we can also loop by accessing values at each index:
>>> for index in range(len(names)):
>>>     print('Hi ' + names[index] + ', your index is ' + str(index))
Hi Ruth, your index is 0
Hi Olivia, your index is 1
```

What is the output?

```python
a = 'hip'
for i in range(1, len(a)):
    print(a[i] + a[i - 1])
```

A. An error

B. hp, ih, pi

C. ih, pi

D. hi, ip

E. I don’t know

Are these valid lists?

```python
a = [5, 'hippo', 3]
b = [1]
```

A. Both are

B. Neither is

C. `a` is but `b` is not

D. `b` is but `a` is not

E. I don’t know

Working with text: strings

We have already seen some strings in the first session

```python
s = 'hippopotamus'
```

We can get parts of a string by **slicing**

```python
>>> print(s[0]) # indexing is 0, 1, 2, ...
h
>>> print(s[1:4])
ipp
>>> print(s[4:9:2])
ooa
>>> print(s[:len(s)-1])
hippopotamu
```

`s[start:end:step]` gets characters from `start` index to `end - 1` index with specified `step` size (which could be negative).

Looping and methods

```
>>> s1 = 'hiphop'
>>> s2 = 'opotamus'
>>> s = s1 + s2
>>> print(s)
hiphopopotamus
>>> for char in s:
>>>     print(char + '!')
```

Methods are functions associated with a type of object, here strings. They are used with the “dot notation”.

```
>>> s = 'I like blackberries.'
>>> new_s = s.replace('black', 'blue')
>>> print(new_s)
I like blueberries.
```

There are many string methods – some useful are ones in the online session materials

Today

1. Recap
2. Lists and `for` loops
3. **Problem solving and debugging**

Problem Solving

Problem: Given a list containing numbers, write a function that returns the number of unique numbers in the list.

Problem Solving

Problem: Given a list containing numbers, write a function that returns the number of unique numbers in the list.

1. Make sure you understand the problem. Try examples.
2. Design a solution. Use paper. Find patterns. Check module materials. Try things out. Solve a part of the problem or a special case.
3. Implement the solution. Start from small parts. If your code does not work, trace what the program does line by line to find where the problem is. Use the print function.
4. Test the solution with examples.

Solution

First: talk to the person next to you. What have you thought about doing? Did they use the same approach?

Solution

First: talk to the person next to you. What have you thought about doing? Did they use the same approach?

No one *best* solution (in a job interview, faster generally better).

One approach: create a new empty list. Loop through items in current list. For each item, if not already in unique_list, then add it to list. Finally, measure size of new list.

Solution

First: talk to the person next to you. What have you thought about doing? Did they use the same approach?

No one *best* solution (in a job interview, faster generally better).

One approach: create a new empty list. Loop through items in current list. For each item, if not already in unique_list, then add it to list. Finally, measure size of new list.

Better approach would be to *sort* the list first (why?).

Solution

First: talk to the person next to you. What have you thought about doing? Did they use the same approach?

No one *best* solution (in a job interview, faster generally better).

One approach: create a new empty list. Loop through items in current list. For each item, if not already in unique_list, then add it to list. Finally, measure size of new list.

Better approach would be to *sort* the list first (why?).

- Answer: after sorting, equal items are adjacent, so you can count uniques in one pass

First Solution in Python

```python
def count_unique_entries(input_list):
    unique_list = []

    for entry in input_list:
        if entry not in unique_list:
            unique_list.append(entry)

    unique_count = len(unique_list)

    return unique_count
```

Problem Solving II

Given two strings, write a method to decide whether one string is a permutation of the other. Use the same strategies as the last problem.

Problem Solving II

Given two strings, write a method to decide whether one string is a permutation of the other. Use the same strategies as the last problem.

Talk to the person next to you: did you both use same strategy?

Problem Solving II

Given two strings, write a method to decide whether one string is a permutation of the other. Use the same strategies as the last problem.

Talk to the person next to you: did you both use same strategy?

Solutions?

- Obvious check: are strings of same length?
- Sort both strings, and see if identical when sorted
    - We'll see sorting later in the class. For now, just know it's an built-in function in Python that we can call.
- Alternatively: we know there are 26 letters in the alphabet. So, define array with 26 entries all equal to 0. When character in first string is in *i*th position, increase *i*th entry by 1. When character in second string is in *i*th position, decrease *i*th entry by 1. Strings match if and only if the final array is all zeroes.

(Code to implement this omitted; exercise)

Problem Solving III

Write a function that given an input *n*, outputs the first *n* Fibonacci numbers. Use the same strategies as last problem.

Problem Solving III

Write a function that given an input *n*, outputs the first *n* Fibonacci numbers. Use the same strategies as last problem.
Talk to the person next to you. How did you get on?

Problem Solving III

Write a function that given an input n, outputs the first n Fibonacci numbers. Use the same strategies as last problem.
Talk to the person next to you. How did you get on?
Solutions:

- You might think you need to handle the base cases where $n = 1$ and $n = 2$ separately.
    - Actually, can treat current as 1 and previous as 0, then no need!

Problem Solving III

Write a function that given an input *n*, outputs the first *n* Fibonacci numbers. Use the same strategies as last problem.
Talk to the person next to you. How did you get on?
Solutions:

- You might think you need to handle the base cases where $n = 1$ and $n = 2$ separately.
    - Actually, can treat current as 1 and previous as 0, then no need!
- Use two integer variables, one that tracks current number, one that tracks prev number, and iteratively print and update.

A Solution in Python

```python
def print_fibonacci(n):
    a = 1 #Current value
    b = 0 #Previous value

    for i in range(n):
        print(a)
        c = a
        a = a + b
        b = c
```

Debugging

Case study: we'd like to design an algorithm that guesses a number between 1 and 100 as quickly as possible. Should work as follows:

- The user guesses a number, and doesn't tell the program.
- The program makes a guess, and the user says 'Higher', 'Lower', or 'Correct'. Higher means that the number is higher than the guess.
- The program stops running when it correctly guesses the number.

We're going to write a Python program to implement this. But the Python code is a bit buggy, so we'll need your help to fix it.

Original Version of Code

```python
def guess_the_number_game():
    print("Think of a number between 1 and 100.")
    lower_limit, upper_limit = 1, 100
    attempts = 0

    while True:
        # Make a guess
        computer_guess = int(lower_limit + upper_limit / 2)
        print(f"My guess is: {computer_guess}")

        user_feedback = input("Is your number 'Higher', 'Lower', or 'Correct'? ")

        # Update the guess range based on user feedback
        if user_feedback == 'higher':
            lower_limit = computer_guess + 1
        elif user_feedback == 'lower':
            upper_limit = computer_guess - 1
        else:
            print(f"Yay! I guessed your number {computer_guess} in {attempts} attempts.")
            break

        attempts += 1

guess_the_number_game()
```

What Are Issues With This Code?

- Attempts is configured incorrectly: If it guesses correctly straight away, it says it finishes after 0 attempts, but it guessed once. Need to move the “attempts+=1” line to right where we guess.

What Are Issues With This Code?

- Attempts is configured incorrectly: If it guesses correctly straight away, it says it finishes after 0 attempts, but it guessed once. Need to move the “attempts+=1” line to right where we guess.
- The user feedback prompt asks for feedback which is capitalized, but accepts responses in lower case only.

What Are Issues With This Code?

- Attempts is configured incorrectly: If it guesses correctly straight away, it says it finishes after 0 attempts, but it guessed once. Need to move the “attempts+=1” line to right where we guess.
- The user feedback prompt asks for feedback which is capitalized, but accepts responses in lower case only.
- Only one of the two limits is divided by two, because of incorrect use of braces.

What Are Issues With This Code?

- Attempts is configured incorrectly: If it guesses correctly straight away, it says it finishes after 0 attempts, but it guessed once. Need to move the “attempts+=1” line to right where we guess.
- The user feedback prompt asks for feedback which is capitalized, but accepts responses in lower case only.
- Only one of the two limits is divided by two, because of incorrect use of braces.
- Meaning of “upper” and “lower” should be other way around!
- We should not assume that in the “else” clause the user has entered “correct” by default

Corrected Version of Code

```python
def guess_the_number_game():
    print("Think of a number between 1 and 100.")
    lower_limit, upper_limit = 1, 100
    attempts = 0

    while True:
        computer_guess = int((lower_limit + upper_limit) / 2)
        print(f"My guess is: {computer_guess}")
        attempts += 1

        user_feedback = input("Is your number 'higher', 'lower', or 'correct'? ").lower()

        # Update the guess range based on user feedback
        if user_feedback == 'higher':
            lower_limit = computer_guess + 1
        elif user_feedback == 'lower':
            upper_limit = computer_guess - 1
        elif user_feedback == 'correct':
            print(f"Yay! I guessed your number {computer_guess} in {attempts} attempts.")
            break
        else:
            print(f"Error")
            break

guess_the_number_game()
```

How Fast is our Method?

We use something called *binary search* to discover secret number.

How Fast is our Method?

We use something called *binary search* to discover secret number.

One can show that this terminates after $log_2(n)$ iterations, where n is the size of the gap between the upper and lower limits. Why?

How Fast is our Method?

We use something called *binary search* to discover secret number.

One can show that this terminates after $log_2(n)$ iterations, where n is the size of the gap between the upper and lower limits. Why?

Each time we make a guess, we partition the space of numbers in 2. After k iterations, we could be in any one of 2^k partitions.

When $2^k \geq n$ we are done, so

$$\min_{k \in \mathbb{N}} k : \; 2^k \geq n \implies k \approx \log_2(n)$$

How Fast is our Method?

We use something called *binary search* to discover secret number.

One can show that this terminates after $log_2(n)$ iterations, where n is the size of the gap between the upper and lower limits. Why?

Each time we make a guess, we partition the space of numbers in 2. After k iterations, we could be in any one of 2^k partitions.

When $2^k \geq n$ we are done, so

$$\min_{k \in \mathbb{N}} k : \; 2^k \geq n \implies k \approx \log_2(n)$$

We'll see more of this next lecture, when we discuss complexity.

Lecture 3: Summary

Today we covered:

- **For loops:** looping over `range(...)` and over items in a sequence.
- **Sequential data: strings and lists**
  - Indexing and slicing: `s[i]`, `s[start:end:step]` (step can be negative).
  - Common patterns: build a new string/list by iterating; loop by value and loop by index.
  - String methods (dot notation), e.g. `replace`.
- **Initial workflow:** understand $\rightarrow$ design $\rightarrow$ implement $\rightarrow$ test.

Up next: complexity theory in week 4

Homework 1: Announcement

Homework 1 will be released later this week, once we finalize it (we will send an announcement once it's ready)

It will be due on 16 Feb at 9am. Two parts to the homework:

- (a) A written part, to be uploaded via Insendi. Like questions you might see in the exam. Worth 5% of your overall grade
- (b) A coding part, to be submitted via Codespaces (like your session exercises). Worth 5% of your overall grade.

Session Work

Session 3 Codespace link in Ed Discussion

- Go through the HTML instructions in `session_3.html`
- At some point, you'll need the `ses03.py`-file with the scaffolded code
- Before you submit, run all the `python ok -q function_name` tests on the command line
- Submit through the Codespace as usual
- If you have time, do the optional exercises in `ses03_extra.py`