CSCA08 AMACSS Review Seminar

Tabeeb Yeamin and Shree Shah

Follow along: https://tinyurl.com/A08FinalSeminar

Dictionaries

```
{key: value}
```

- Keys must be an immutable type, like string, int, tuple etc.
- `{[1,2,3]: 'a'}` will fail.
- Value can be any type, can also be lists, sets, dicts etc.
- `{'a': [1,2,3], 'b': {1:(2,2,2)}}`

Dictionaries

Initialization:

- `d = {}`
- `d = dict()`

initializing set => `set()`

Dictionaries are NOT ordered! (Just like sets)

- cannot do d[1]

Dictionaries

```
name_to_age = {'Alice': 30, 'Bob': 15, 'Carol': 45,
'David': 25, 'Ed': 35}
```

Access the values by using [key] or .get(key) method:

- `name_to_age['Alice']` => 30 — invalid key gives error
- `name_to_age.get('Alice')` => 30 — invalid key returns None
- `name_to_age.get('Alice', -1)` => 30 — invalid key returns -1

Dictionaries

```
name_to_age = {'Alice': 30, 'Bob': 15, 'Carol': 45,
'David': 25, 'Ed': 35}
```

- You can use elemental for loops to loop through the keys

  ```
  for name in name_to_age:
      print(name_to_age[name])
  ```
  <= prints the ages

- You can also get list of the keys by doing:

  ```
  name_to_age.keys()
  ```

  Returns dict keys object, a “list-like” object, but it does not support indexing.

Files

`open(filename, mode)` : a function that returns a filehandle object

(str, str) -> (io.TextIOWrapper)

mode:

- `'r': reading`
- `'w': writing (erases previous data)`
- `'a': appending`

Close the filehandle object by doing:

```
filehandle.close()
```

Files (Reading)

`filehandle.readline()` : 1 line from the file

`filehandle.read()`: reads whole file into a single string

`filehandle.readlines()`: reads whole file into a list (each element is one line of text)

Files Example

HBD.txt has:

```
HAPPY
BIRTHDAY
TO
YOU
```

```
filehandle.readline()
```

```
filehandle.read()
```

```
filehandle.readlines()
```

Files Example

HBD.txt has:

```
HAPPY
BIRTHDAY
TO
YOU
```

```
filehandle.readline()
```

=> “HAPPY\n” (the \n will create a new line)

```
filehandle.read()
```

```
filehandle.readlines()
```

Files Example

HBD.txt has:

```
HAPPY
BIRTHDAY
TO
YOU
```

```
filehandle.readline()
```

=> “HAPPY\n” (the \n will create a new line)

```
filehandle.read()
```

=> “HAPPY\n BIRTHDAY\nTO\nYOU”

```
filehandle.readlines()
```

Files Example

HBD.txt has:

```
HAPPY
BIRTHDAY
TO
YOU
```

```
filehandle.readline()
```

=> “HAPPY\n” (the \n will create a new line)

```
filehandle.read()
```

=> “HAPPY\n BIRTHDAY\nTO\nYOU”

```
filehandle.readlines()
```

=> [“HAPPY\n”, “BIRTHDAY\n”, “TO\n”, “YOU”]

Looping through a file

HBD.txt has:

```
HAPPY
BIRTHDAY
TO
YOU
```

```
for line in filehandle:

    print(line)
```

Looping through a file

HBD.txt has:

```
HAPPY
BIRTHDAY
TO
YOU
```

```
for line in filehandle:

    print(line)
```

“HAPPY”

“BIRTHDAY”

“TO”

“YOU”

Looping through a file

HBD.txt has:

```
HAPPY
BIRTHDAY
TO
YOU
---
```

```
for line in filehandle:
    print(line)
```

“HAPPY”

“BIRTHDAY”

“TO”

“YOU”

“---”

Looping through a file

HBD.txt has:

```
HAPPY
BIRTHDAY
TO
YOU
---
```

```
line = filehandle.readline()

while (not line.startswith("---")):

  print(line)

  line = filehandle.readline()
```

"HAPPY"

"BIRTHDAY"

"TO"

"YOU"

Files (Writing)

`filehandle.write(text)`:

- Only takes in strings
- To create new lines you must include `"\n"`
- You can also use `"\t"` to tab
- Make sure to close your file afterwards, or it may not write

File + Dictionaries Example (2017 A08 TT2)

Brian built some tools to work with grade files. The files consist of a name, a course and a grade separated by commas, one grade per line. After the grade data is a line starting with --- and then other data. A sample file might look something like the following:

```
Alice,CSCA08,99
Bob,CSCA08,70
Alice,MATA31,95
Alice,CSCA48,85
Carol,ABCA01,60
Bob,CSCA48,50
---
This file is private and confidential...
```

Brian wrote a function called `build_marks_dict` that reads a grade file and turns it into a dictionary that maps student names to dictionaries mapping courses to grades. A sample dictionary of that type might look something like:

```
{'Alice': {'CSCA08': 99.0, 'MATA31': 95.0, 'CSCA48': 85.0},
 'Bob': {'CSCA08': 70.0, 'CSCA48': 50.0},
 'Carol': {'ABCA01': 60.0}
}
```

```python
def build_marks_dict(input_file):
    # create an empty student to marks dictionary

    # read a line to start with

    # loop through the input as long as it doesn't start with ---

        # strip the input line (bc there is going to be an extraneous \n)

        # get the student, course, grade data

        # create an empty course to grade

        # if student in the student to marks dict

            # get the student's grades (remember, dicts are mutable)

        # otherwise

            # add the new course to grade info for the student

        # add the new grade to the students existing grades

        # read the next line

    return student_to_marks
```

```python
def build_marks_dict(input_file):
    # create an empty student to marks dictionary
    student_to_marks = {}
    # read a line to start with
    input_line = input_file.readline()
    # loop through the input as long as it doesn't start with ---
    while (not input_line.startswith("---")):
        # strip the input line (bc there is going to be an extraneous \n)
        input_line = input_line.strip()
        # get the student, course, grade data
        (student, course, grade) = input_line.split(',')
        # create an empty course to grade
        course_to_grade = {}
        # if student in the student to marks dict
        if (student in student_to_marks):
            # get the student's grades (remember, dicts are mutable)
            course_to_grade = student_to_marks[student]
        # otherwise
        else:
            # add the new course to grade info for the student
            student_to_marks[student] = course_to_grade
        # add the new grade to the students existing grades
        course_to_grade[course] = float(grade)
        # read the next line
        input_line = input_file.readline()
    return student_to_marks
```

UnitTesting (Number Ranges)

Ex: 0 <= n <= 13

Test the edge cases, a number in between, and then above and below the range

- n = 0
- n = 13
- 0 < n < 13
- n > 13
- n < 0 (If valid input)

You don’t need to test invalid cases. I.e. if n refers to age, n >= 0 should be a REQ.

UnitTesting Ex 1 (April 2017 Final)

In Canada, the Federal and Provincial goverments uses a progressive tax system. For many Canadians, this means that when their income goes up, their tax rate goes up too. Marginal income tax rates are used when determining the total amount of tax due. The 2016 Federal marginal income tax rates in Canada are given in the following table:

bracket 1 up to $45,282	bracket 2 over $45,282 up to $90,563	bracket 3 over $90,563 up to $140,388	bracket 4 over $140,388 up to $200,000	bracket 5 over $200,000
15%	20.5%	26%	29%	33%

```
def get_marginal_tax(income):
    """ (float) -> float

    Precondition: income >= 0.
```

```
>>> get_marginal_tax(0)
0.15
>>> get_marginal_tax(1165701.85)
0.33
"""
```

UnitTesting Ex 1 (April 2017 Final)

Test Case Description	Income ($)	Return value
0	0	0.15
In bracket 1	20, 000	0.15
Bracket 1 upper edge case	45, 282	0.15
In bracket 2	60, 000	0.205
Bracket 2 upper edge case	90, 563	0.205
In bracket 3	110, 000	0.26
Bracket 3 upper edge case	140, 388	0.26
In bracket 4	160, 000	0.29
Bracket 4 upper edge case	200, 000	0.29
Bracket 5	201, 000	0.33

UnitTesting Strings/Lists/Dicts etc.

- Empty
- One character/element
- More than one character/element (May have to divide further depending on the question)

UnitTesting Strings Example

```
def is_palindrome(string: str) -> bool:
    ''' Returns True iff string is a palindrome
    Precondition: 0 <= len(string) <= 3
    '''
```

UnitTesting Strings Example

len(string)	strings
0	“”
1	“a”
2	“aa”, “ab”
3	“aaa”, “aba”, “abb”, “aab”, “abc”

PCRS Exam Review Question

In this problem, you will write a function `contains_no_unique_hashtags` that takes two lists as parameters. The first list, `tweet_hashtags`, contains lists. Each list represents the hashtags found in a single tweet. The second list, `unique_hashtags`, is a list of hashtags uniquely used by a specific candidate.

`contains_no_unique_hashtags` should return a new list of the hashtag lists for those tweets (from the original list) that **cannot** be attributed to the candidate. A tweet cannot be attributed to the candidate if it uses none of the unique hashtags used by the candidate.

Your function should not change the original lists passed as parameters.

```python
def contains_no_new_hashtags(tweet_hashtags: List[list[str]], unique_hashtags: List[str]) -> List[list[str]]
    """ (list of list of str, list of str) -> list of list of str

    >>>contains_no_unique_hashtags([["#hello", "#a08"], ["#wow"], ["#spectacular", "#summer"]], ["#hello"])
    [["#wow"], ["#spectacular", "#summer"]]

    >>>contains_no_unique_hashtags([["#hello", "#a08"], ["#wow"], ["#cool", "#summer"]], ["#hello", "#cool", "#wow"])
    []
    """
    no_candidate=[]

    #loop through each sublist in tweet_hashtags
    for tweet in tweet_hashtags:
        #set initiallly that the sublist in questions belongs to some candidate
        unique=True
        #loop through each hashtag in sublist
        for hashtag in tweet:
            #if the hashtag is in unique_hashtags, it belongs to a candidate
            if hashtag in unique_hashtags:
                unique=False
                break           #disgard rest of hashtags

        #if the sublist was unique, i.e if it belonged to no candiate, append to return list
        if unique:
            no_candidate.append(tweet)
```

DO WE NEED THE BREAK?

Removing it does not change the code, but it allows us to avoid checking hashtags after we have already established that the specific list is not to be added.

It helps specifically, in terms of complexity and time. Consider a sublist of length 50 and the first word indicating that we need not consider this list. No break will indicate unnecessary looping.

Basics Overview - Runtime

Best Case Scenario

the term is used to figure out what the least amount of iterations an algorithm could possibly take [the ideal input]

Worst case Runtime

the term is used to figure out what the least amount of iterations an algorithm could possibly take [input a programming would not want to see]

Question - Does every algorithm need to have a worst case and best case?

No, some algorithms are not affected by content of the input, only by size of input. Examples: bubble sort, finding max/min value, reversing a string

Algorithm Runtime

CONSTANT

Algorithm performance is the same regardless of the size of the input

As size of list or item in concern changes, number of iterations do not.

General example:
Range

LINEAR

Algorithm performance is proportionate to size of input

As size of list or item in concern changes, number of iterations changes at the same rate

General example:
for item in list

QUADRATIC

Algorithm performance is proportionate to the square of the size of input

As size of list or item in concern changes, number of iterations changes by 2 fold

General example:
Nested for loops

LOGARITHMIC

Algorithm performance is proportionate to log base 2 of the size of input.

Number of iterations only changes when there size of input is doubled

General example:
Binary Search

Runtime Practice

```
for i in range(10):
    for j in range(20):
        for p in range(len(mylist)):
            #do something interesting
```

What is the runtime complexity of the algorithm:

(a) Constant (b) Quadratic (c) Linear (d) Logarithmic

```
for i in range(1, len(mylist), 2):
    for i in range(10):
        print(str(i) + ",")
```

mylist refers to a list. If there are k items in the list, roughly how many iterations are there?

(a) k (b) k^2 (c) 20 (d) 10k (e) 10k/2

```
total=0
for sublist in mylist:
    for item in sublist:
        total=total+item
```

What is the runtime complexity of the algorithm:

(a) Constant (b) Quadratic (c) Linear (d) Logarithmic

Winter 2018 Question

Question 6. [6 MARKS]

Answer the questions below about the following function:

```
def is_level(L: List[int]) -> bool:
    """Return True if and only if there is no gap larger than 1 between any
    two adjacent numbers in L.

    >>> is_level([1, 3])
    False
    >>> is_level([1, 2, 3, 4, 5, 4, 5])
    True
    >>> is_level([1, 2, 3, 2, 3, 1])
    False
    """
    for i in range(len(L) - 1):
        if abs(L[i] - L[i + 1]) > 1:
            return False
    return True
```

Let k be the number of elements in L.

Give a formula in terms of k to describe exactly how many comparisons happen in is_level in the best case.

```
1
```

Briefly describe the property of L that causes the best case for is_level.

```
First pair of items (index 0 and 1)
have a gap greater than 1

Example: [1, 5, 6, 8]
```

What best describes the best case running time of is_level.

```
Constant
```

Give a formula in terms of k to describe exactly how many comparisons happen in is_level in the worst case.

```
k-1
```

Briefly describe the property of L that causes the worst case for is_level

.

```
Every pair of adjacent numbers has a
gap of ≤1 OR last pair has a gap
greater than 1
```

What best describes the worst case running time of is_level.

```
Linear
```

Fall 2015 Question

Consider the function bogosort(list). This function shuffles an input list randomly, until it is sorted.

```
import random

def bogosort(lst):
    """ (list of int) -> NoneType

    Modify lst to sort the items from smallest to largest.

    >>> my_list = [42, 17, 56]
    >>> bogosort(my_list)
    >>> my_list
    [17, 42, 56]
    """

    while not is_sorted_list(lst):
        random.shuffle(lst)
```

Task 1: Complete function Is_sorted_list(lst)

```
def is_sorted_list(lst):
    """ (list of int) -> bool

    Return True if and only if the items in lst are sorted from smallest to largest.

    >>> is_sorted_list([12, 12, 2015])
    True
    >>> is_sorted_list([11, 1, 2016])
    False
    """

    for i in range(1,len(lst)):
        if lst[i] < lst[i-1]:
            return False
    return True
```

```python
def is_sorted_list(lst):
    """ (list of int) -> bool

    Return True if and only if the items in lst are sorted from smallest to largest.

    >>> is_sorted_list([12, 12, 2015])
    True
    >>> is_sorted_list([11, 1, 2016])
    False
    """


    for i in range(1,len(lst)):
        if lst[i] < lst[i-1]:
            return False
    return True
```

Size 4 input to achieve best case for is_sorted_list

9 4 2 1
(decreasing order)

Word that best describes best case scenario for is_sorted_list

Constant (1)

Size 4 input to achieve worst case for is_sorted_list

1 5 7 9
(increasing order)

Word that best describes worst case scenario for is_sorted_list

Linear

Sorting Algorithms - Explained

BUBBLE SORT

Compares and swaps adjacent items so that in the i'th pass, i elements at the end of the list will be sorted.

Simply put, bubbles the largest element from index 0 to end.

Complexity: Quadratic
(in all cases)

INSERTION SORT

Starts at the beginning of the list, moves towards the end, and inserts the i'th element in the correct position in the sorted part of the list.

Complexity:
Best - Linear
Worst - Quadratic

SELECTION SORT

Finds the smallest value in the list from index i to the length of the list, and swaps the smallest value with the value at the i'th index.

Index i is the first element in the unsorted part of the list

Complexity: Quadratic
(in all cases)

Bubble Sort Example

List - [3, 2, 9, 10, 4]

Pass: 1

[3, 2, 9, 10, 4] → [2, 3, 9, 10, 4] → [2, 3, 9, 10, 4] → [2, 3, 9, 10, 4] → [2, 3, 9, 4, 10]

Pass: 2

[2, 3, 9, 4, 10] → [2, 3, 9, 4, 10] → [2, 3, 9, 4, 10] → [2, 3, 4, 9, 10]

Pass: 3

[2, 3, 4, 9, 10] → [2, 3, 4, 9, 10] → [2, 3, 4, 9, 10]

Pass: 4

[2, 3, 4, 9, 10]→ [2, 3, 4, 9, 10]

Pass: 5

2, 3, 4, 9, 10]

The pink highlighted represents the sorted portion of the list after each pass

Yellow text is comparison in that iteration

Selection Sort Example

List - [3, 2, 9, 10, 4]

Pass: 1 (i=0)
[3, 2, 9, 10, 4] → [2, 3, 9, 10, 4]
Pass: 2 (i=1)
[2, 3, 9, 10, 4] → [2, 3, 9, 10, 4]
Pass: 3 (i=2)
[2, 3, 9, 10, 4] → [2, 3, 4, 10, 9]
Pass: 4 (i=3)
[2, 3, 4, 10, 9] → [2, 3, 4, 9, 10]
Pass: 5 (i=4)
[2, 3, 4, 9, 10] → [2, 3, 4, 9, 10]

The pink highlighted represents the sorted portion of the list after each pass

The yellow text is the smallest value in the Range from index i (inclusive) to len(list) (exclusive)

The purple text is the i'th element, the element that will be swapped with the smallest value in range(i, len(list))

Insertion Sort - Best Case

List - [3, 5, 9, 10, 41]

Pass: 1

[3, 5, 9, 10, 41] → [3, 5, 9, 10, 41]

Pass: 2

[3, 5, 9, 10, 41] → [3, 5, 9, 10, 41]

Pass: 3

[3, 5, 9, 10, 41] → [3, 5, 9, 10, 41]

Pass: 4

[3, 5, 9, 10, 41] → [3, 5, 9, 10, 41]

Pass: 5

[3, 5, 9, 10, 41] → [3, 5, 9, 10, 41]

Best case is linear runtime because the i'th element is always in the correct position. In the following code where the shifts are to take place, the while loop never runs because list[i-1]>value is never met.

Only one comparison takes place, after which the while loop condition fails (comparison in the yellow text)

```
while i > 0 and lst[i - 1] > value:
    lst[i] = lst[i - 1]
    i = i - 1
```

Insertion Sort - Worst Case

List - [41, 10, 9, 5, 3]

The yellow pairs indicate the comparisons which accumulate to the amount of shifting that must take place. Shifting code shown in image

In the i'th iteration, i shifts must take place to put the i'th element into the correct spot (beginning of the list)

```
while i > 0 and lst[i - 1] > value:
    lst[i] = lst[i - 1]
    i = i - 1
```

Pass: 1 (i=0)

[41, 10, 9, 5, 3]

Pass: 2 (i=1)

[41, 10, 9, 5, 3] → [10, 41, 9, 5, 3]

Pass: 3 (i=2)

[10, 41, 9, 5, 3] → [10, 41, 9, 5, 3] →[9, 10, 41, 5, 3]

Pass: 4 (i=3)

[9, 10, 41, 5, 3] → [9, 10, 41, 5, 3] → [9, 10, 41, 5, 3] → [5, 9, 10, 41, 3]

Pass: 5 (i=4)

[9, 10, 41, 5, 3] → [9, 10, 41, 5, 3] → [9, 10, 41, 5, 3] → [9, 10, 41, 5, 3] → [3, 5, 9, 10, 41]

Questions?