CSC110 Leture 2: Representing Data

Hisbaan Noorani

September 14, 2020

Contents

1 Ex 1: Categorizing Data

Imagine that you are a part of a deeply interconnected society of big cats owners in America. In particular, you are the proud owner of a private zoo set up in Oklahoma that houses many species of dangerous big cats such as tigers, lions, puma, ligers and tigons. You know of other owners of private zoos and their "collections" of large cats.

1. Think about different kinds of data you would have in this setting. For each data type below, write a description of a piece of data that would have that type. We have completed the first row for you.

Data Type	Example
Boolean	"Have I been shut down by the government today?"
Natural Number	"The number of big cats in my posession"
Real Number	"The ratio of big cats to owners where it is <owners>/<big cats>"
String	"The name of one of the big cats"
Set	"The set of names of the big cats"
List	"The set of names of the big cats, arranged by aquisition date"
Map	"The names of the big cats mapped to their ages"

2 Ex 2: Data Types in Python

1. In the table below, write down the python type for each python literal. The first row is done for you.

Data Type	Python Literal
int	12345
float	12345.0
bool	True
str	'You are doing great :)'
set	{'David', 'Mario'}
list	[1, 2, 3, 4, 5]
tupple	('wow', 'roud', 'brackets')
dictionary	{'cool'}

2. What is the limitation of the `float` datatype, and why does this limitation exist?

   Floats are limited to a certain amount of decimal places because computers have a finite amount of ram.

3. Without using the python console, complete the table below.

Python Expression	Value	Type of Value
9 / 3	3.0	float
9 // 3	3	int
2 + 2.0	4.0	float
3 int {1, 2}	False	bool
'x' + 'y'	'xy'	str
not False	True	bool
\[1, 2] + [2, 3]	[1, 2, 2, 3]	list
\['hi', 'bye'][0]	'hi'	str

4. In the table below, add parenteses to indicate the order that operations are evaluated (the last one is not obvious)

Python Expression	Expression with ()	Type of result
3 + 5 * 2	3 + (5 * 2)	int
4 + 8 / 2 ** 2 / -2	4 + ((8 / (2 ** 2)) / 2)	int
-2 + 15 % 7	-2 + (15 % 7)	int

3 Ex 3: Variables

1. Consider this code

```
>>> x = 4
>>> y = x + 2
>>> z = {y: x}
```

   After the above code is executed, to which values do `x`, `y`, and `z` refer?

Variable	Value
x	4
y	6
z	{6: 4}

2. Determine whether the assignemnt statements below have been properly expressed. If it is, move onto the next statement. If it is not, explain why it is incorrect and correct the statemet.

   (a) `5 = x`

   Variable names cannot begin with a number should be: x = 5

   (b) `x = 5.0`

   Correct

   (c) `x = hello!`

   If hello! is a string, it must be surrounded by " If hello! is a variable name, it cannot contain ! should be either x = 'hello!' or x = hello

   (d) `x = true`

   True should be capitalized if it is a boolean or surrounded by " if it is a string should be either x = True or x = 'true'

   (e) `x = set()`

   Correct

   (f) `x = [1, 'hi', 3]`

   Correct

   (g) `x = {1: 'two', 'three'}`

   dict incomplete should be x = {1: 'two', 2: 'three'}

4 Ex 4: comprehensions and range

1. What range of numbers do the following python range expressions represent?

   (a) `range(0, 5)`
   0, 1, 2, 3, 4

   (b) `range(5, 10)`
   5, 6, 7, 8, 9

   (c) `range (5, 5)`
   []

2. What is one similarity between a set literal and a set comprehension? What is one difference?

   Similarity - They both use curly braces {} to define them.

   Difference - Set literals are fixed, set comprehension can vary based on variables.

3. For each of the following descriptions, write a comprehension that evalueates to the set.

   (a) The set of integers between 30 and 50, inclusive.

   ```
   1  {v for v in range(30, 51)}
   ```

   (b) The set of squares of the natural numbers less than 2000.

   ```
   1  {v ** 2 for v in range(0, 44)}
   ```

   (c) The list of integers between -30 and 30 inclusive.

   ```
   1  [v for v in range(-30, 31)]
   ```

4. You are given a variable s that regers to a (very very long string). Write an expression that evaluates to a list containing the first 20 characters in the string, in the order they appear.

```
1  [s[v] for v in range(20)]
```

5. For each of the following mappings, write a Python dictionary expression that evaluates to the mapping.

   (a) A mapping from a number to its square for the first 50 natural numbers.

```
{v: v ** 2 for v in range(51)}
```

   (b) A mappint from input to ouput of the functoin f(x) = x/(x-1) for integer inputs greater than 1 and lesst than 2000

```
{v: (v)/(v-1) for v in range (2,2000)}
```

   (c) A mapping from a number to a list containing that many elements of the string 'Hello' for the first 40 natural numbers

```
{v: {'Hello' for x in range v} for v in range(51)}
```

   (d) A mapping from an integer to the set of integers between 0 and that integer, for integers from 1 to 20, inclusive

```
{v: {x for x in range(v + 1)} for v in range(21)}
```

CSC110 Lecture 3: Functions

Hisbaan Noorani

September 15, 2020

Contents

1 Ex 1: Practice with built-in functions

1. Suppose we have executed the following assignment statements in the python console:

```
>>> n = -5
>>> numbers_list = [1, 10, n]
>>> numbers_set = {100, n, 200}
```

Write down what each of the following expressions evalusate to. Do this by hand first! (Then check your work in the python console.)

```
>>> abs(n)
5

>>> sorted(numbers_list)
[-5, 1, 10]

>>> sorted(numbers_set) + sorted(numbers_list)
[-5, 100, 200, -5, 1, 10]

>>> type(numbers_set)
<class 'set'>

>>>type(numbers_list == n)
<class 'bool'>

>>> sum(numbers_set) - n
300

>>> max(numbers_list + [5])
10
```

2 Ex 2: Practice with methods

Note: you might want to use Section A.2 Python Build-in Data Types REference as a reference in this exercise.

1. Suppose we have executed the following assignment statemnt in the Python console:

```
>>> wish = 'Happy Birthday'
```

   Write down what each of the following expressions evaluate to. Do this by hand first! (Then check your work in the python console.)

```
>>> str.lower(wish)
'happy birthday'

>>> str.lower(wish[0]) + str.lower(wish[6])
'hb'

>>> str.isalnum(wish)
False

>>> str.count(wish, 'y')
2
```

2. Suppose we have executed the following assignment statments in the Python console:

```
>>> set1 = {1, 2, 3}
>>> set2 = {2, 4, 5, 10}
>>> set3 = {'cat', 'dog', 'rabbit'}
```

   Write down what each of the following expressions evaluate to. Don't worry about the order of elements in a set. Do this by hand first! (Then check your work in the Python console.)

```
>>> set.union(set1, set2)
{1, 2, 3, 4, 5, 10}

>>> set.intersection(set1, set2)
{2}

>>> set.intersection(set1, set3)
{}

>>> set.difference(set3, set2)
{'cat', 'dog', 'rabbit'}

>>> set.difference(set1, set2)
{1, 3}

>>> set.symmetric_difference(set1, set2)
{1, 3, 4, 5, 10}
```

3 Ex 3: Functoins definitons in Python

1. Answer the questions below about the following function definiton.

```
def calculate(x: int, y: int) -> list:
    """Return a list containing the sum and product of the two given numbers.
    """
    return [x + y, x * y]
```

(a) What is the function header?
def calculate(x: int, y: int) -> list:

(b) What is the function name?
calculate

(c) How many parameters does this function have? What are their names and types?
2, x -> int, y -> int

(d) What is the function's return type?
list

(e) What isthe part surrounded by tripple-quotes (“”“) called? What is it's purpose?
Documentation. It describes what the function does.

(f) What is the function body?
return [x + y, x * y]

(g) Compared to the examples we looked at in lectures, what part is missing from this function definiton?
Examples in the documentation.

(h) Write down what you would add to complete this funciton definiton.

```
    >>> calculate(0, 0)
    [0, 0]

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

2. For each of the function definitons given below, complete the definition by writing a description and one doctest example.

```
def is_same_length(list1: list, list2: list) -> bool:
    """Return whether both lists are the same length.
    >>> is_same_length([1, 2, 3, 4, 5], [1, 2, 3, 4])
    False
    >>> is_same_length([1, 2, 3], [3, 3, 3])
    True
    """
    return len(list1) == len(list2)

def multiply_sums(set1: set, set2: set) -> int:
    """Return an integer that is equal to the product of the sum of the sets.
    >>> multiply_sums({1, 2}, {3, 4})
    21
    """
    return sum(set1) * sum(set2)

```

```
    def exponentiate(nums: list) -> list:
        """Return the square of every element in the list.
        >>> exponentiate([1, 2, 7])
        [1, 4, 59]
        """
        return [x ** x for x in nums]
```

3. Complete each of the following functions according to their docstring description and doctests.

```
    def different_sums(set1: set, set2: set) -> bool:
        """Return whether set1 and set2 have different sums.
        >>> different_sums({1, 2, 3}, {5, -1})
        True
        >>> different_sums({3}, {1, 2})
        False
        """
        return sum(set1) != sum(set2)
```

```
    def squares(n: int) -> dict:
        """Return a dictionary mapping the numbers from 1 to n to their squares.
        Assume that n > 1.
        >>> squares(3)
        {1: 1, 2: 4, 3: 9}
        """
        return {x: x ** 2 for x in range(n)}
```

4 Ex 4: Function scope and vairables

1. Consider the following code snippet:

```
def add_together(x: int, y: int) -> int:
    return x + y


eleven = add_together(5, 6)
twelve = x + 7
```

Running the code snippets produces the following output:

```
Traceback (most recent call last):
    File "path/to/example-name-error-1.py", line 6, in <module>
        twelve = x + 7
NameError: name 'x' is not defined
```

(a) Which line of code is causing the error?
`twelve = x + 7`

(b) What does `NameError: Name 'x' is not defined` mean?
x is not in the scope where it is called. It exists only inside of the function.

(c) Why is `x` not defined?
`x` only exsits within the function `add_together`. It is not accessable from outside of that.

2. Recall the functions square and `calculate_distance`, which some "markers" added:

```
    def square(x: float) -> float:
        """Return x squared.

        >>> square(3.0)
        9.0
        >>> square(2.5)
        6.25
        """
        # MARKER A
        return x ** 2
```

```
    def calculate_distance(p1: tuple, p2: tuple) -> float:
        """Return the distance between points p1 and p2.

        p1 and p2 are tuples of the form (x, y), where the x- and y-coordinates are points.

        >>> calculate_distance((0, 0), (3.0, 4.0))
        5.0
        """
        x1 = p1[0]
        y1 = p1[1]
        x2 = p2[0]
        y2 = p2[1]
        # MARKER B
        return (square(x1 - x2) + square(y1 - y2)) ** 0.5
```

Suppose we run the following in the python conose:

```
    >>> p1 = (0, 0)
    >>> p2 = (5, 3)
    >>> calculate_distance(p1, p2)
```

(a) Draw a value-based memory model diagram to show the current state of the variables when "MARKER B" is reached. Make sure to show both the vriables in the console (`__main__`) and the variables in `calculate_distance`.

`console`:

Variable	Value
p1	(0, 0)
p2	(5, 3)

`calculate_distance`:

Vairable	Value
p1	(0, 0)
p2	(5, 3)
x1	0
y1	0
x2	5
y2	3

`square - first call:`

Variable	Value
x	-5

`square - second call:`

Vairable	Value
x	2

CSC110 Lecture 4: Function Design

Hisbaan Noorani

September 21, 2020

Contents

1 Ex 1: Function design practice

1. Consider the code nippets below. For each code snippet, complete the docstring and/or docstring examples. Use the type contracts, functions name, parameter names, and function body to help you.

```python
def check_lengths(strings: list, max_length: int) -> bool:
    """Return whether all the strings in the set are shorter than the provided max_length

    >>> check_lengths(['cat', 'no', 'maybe'], 4)
    False
    >>> check_lengths(['hello', 'my', 'name', 'is', 'Hisbaan'], 8)
    True
    """
    return max([len(x) for x in strings]) < max_length


def string_lengths(strings: list) -> dict:
    """Return a dictionary mapping a string to it's length

    >>> string_lengths(['aaa', 'david'])
    {'aaa': 3, 'david': 5}
    >>> string_lengths(['one', 'two', 'three'])
    {'one': 3, 'two': 3, 'three': 4}
    """
    return {s: len(s) for s in strings}
```

2. For each of the following descriptions, write a Python function to perform the task described, using the Function Design Recipe.

   (a) Given a `float` representing the price of a product and another `float` representing a tax rate (e.g., a 13% tax rate represented as the `float` value `0.13`), calculate the after-tax cost of the product.

```python
def calculate_item_cost(price: float, tax_rate: float) -> float:
    """Return the price of an object after a tax is applied

```

```
    >>> price_with_tax(10.0, 0.13)
    11.3
    """
    return price + (price * tax_rate)
```

(b) Given a dictionary mapping names of products (as strings) to prices (as floats), which represents a customer order at a store, and a tax rate, calculate the total after-tax cost of the products.

Hint: the following illustrates how to use a dictionary as the "collection" in a comprehension.

```
>>> mapping = {1: 'a', 2: 'b', 3: 'c'}
>>> [k for k in mapping]  # Variable k is assigned to each keys in mapping
[1, 2, 3]
>>> [mapping[k] for k in mapping]
['a', 'b', 'c']
```

```
def calculate_order_cost(prices: dict, tax_rate: float) -> float:
    """Return the total cost of all items in order after tax.

    >>> prices_after_tax({'milk': 3.99, 'eggs': 4.99, 'chips': 2.99}, 0.13)
    13.5261
    """
    return calculate_item_cost(sum(prices[item] for item in prices), tax_rate)
```

2 Ex 2: Function design practice, `math` edition

1. Use the Function Design Recipe to implement the following function: Given three side lengths of a triangle (as floats), calculate the angles in the triangle.

   Include an `import math` statement at the top of your Python file so that you can use definitions from math in your function implementation for this question.

   Hints:

   - The Cosine Law from trigonometry states that for a triangle with side lengths , , and , with angle opposite the side with length , the following equality holds:

   $$c^2 = a^2 + b^2 - 2ab\cos\theta$$

   - The sum of all angles in a triangle add up to 180 degrees, or Π
   - The `math` module has functions both for calculating the trignometric functions and the inverses: `sin` and `asin` (short for "arcsin"), `cos` and `acos`, etc.
   - It also has a variable `pi` you can access in your code as `math.pi`.

```
import math

def calculate_angles(side_a: float, side_b: float, side_c: float) -> list:
    """Return the angles of a triangle based on the given side lengths, a b and c.

    >>> calculate_angles(1, 1, math.sqrt(2))
    {45, 45, 90}
    """
    a_2 = side_a ** 2
```

```
    b_2 = side_b ** 2
    c_2 = side_c ** 2

    angle_a = math.acos((b_2 + c_2 - a_2) / (2 * side_b * side_c))
    angle_b = math.acos((a_2 + c_2 - b_2) / (2 * side_a * side_c))
    angle_c = math.acos((a_2 + b_2 - c_2) / (2 * side_a * side_b))
    return [angle_a, angle_b, angle_c]
```

3 Additional Exercises

1. Given a set of strings, calculate the length of the longest string.

```
def get_longest_string(strings: set) -> int:
    """Return the length of the longest string in the set.

    >>> get_longest_string({'hello', 'my', 'name', 'is', 'hisbaan'})
    7
    """
    return max({len(x) for x in strings)})
```

2. You are renting a car to make a road trip accross Canada. The car rental company you plan to use charges a fee of $50 plus $15 per day you rent the car.

   Given the starting and ending dates of your trip (represented in Python as `datetime.date` values), calculate the total cost of renting a car. (**Note**: the start and end date are both counted in the rental cost.)

   For this function, you should read Section 2.4 Importing Modules for more inforamtion about the `datetime` module.

Lecture 05 Exercises

Hisbaan Noorani

September 22, 2020

Contents

1 Ex 1: Propositional and Predicate Logic

1. Consider the following propositional statement:

   $(p \Rightarrow q) \vee r) \Leftrightarrow (p \Rightarrow (q \wedge r))$

   Complete the following truth table below to practice evaluating larger propositonal statements

p	q	r	$p \Rightarrow q$	$(p \Rightarrow q) \wedge r$	$q \wedge r$	$p \Rightarrow (q \wedge r)$	$((p \Rightarrow q) \wedge r) \Leftrightarrow (p \Rightarrow (q \wedge r))$
F	F	F	T	F	F	T	F
F	F	T	T	T	F	T	T
F	T	F	T	F	F	T	F
F	T	T	T	T	T	T	T
T	F	F	F	F	F	F	T
T	F	T	F	F	F	F	T
T	T	F	T	F	F	F	T
T	T	T	T	T	T	T	T

2. Write a Python function that, given three boolean values p, q, and ,r returns the values of the above propositional statement for those values. We have begun the Function Design Recipe for you.

```
def propositional_formula(p: bool, q: bool, r: bool) -> bool:
    """Return the value of ((p  q)  r)  (p  (q  r)).

    >>> propositional_formula(True, False, False)
    True
    >>> propositional_formula(False, False, False)
    False
    """
    return (((not p) or q) and r) == ((not p) or (q and r))
```

3. Let p be a propositonal variable representing the statement “All cats are cute”, and q be a propostional variable representing the statement “All dogs are cute”.

   (a) Translate the statement $p \Rightarrow q$ into English.

   If all cats are cute, then all dogs are cute.

(b) What is the hypothesis of this implication?
All cats are cute.

(c) What is the conclusion of this implication?
All dogs are cute.

(d) Write down an English translation of the converse of this implication.
If all dogs are cute, then all cats are cute.

(e) Write down an English translation of the contrapositive of this implication.
if all dogs are not cute, then all cats are not cute.

4. Suppose we have a set of computer programs that are each meant to solve the same task. Some of the programs are written in the Python programming language, and some of the programs are written in a programming language that is not Python. Some of the programs correctly solve the task, and others do not.

   Let's define the following predicates:

   $Python(x) : x$ is a Python program, where $x \in P$

   $Correct(x) : x$ solves the task correctly, where $x \in P$

   Translate each statement below from English into predicate logic, or vice versa.

   (a) Program `my_prog` is correct and is written in Python. (`my_prog` is an element of P.
   $Python(\texttt{my\_prog}) \wedge Correct(\texttt{my\_prog})$

   (b) At least on incorrect program is written in Python
   $\exists x \in P : Python(x) \wedge \neg Correct(x)$

   (c) $\exists x \in P, Python(x) \wedge Correct(x)$
   At least one correct program is written in python.

   (d) $\forall x \in P, \neg Python(x) \wedge Correct(x)$
   Every program that is not written in python is correct.

2 Ex 2: Translating statements and filtering collection

1. Using the same set and predicates as the previous question, translate each statement below from English into predicate logic, or vice versa.

   (a) No python program is correct.
   $\forall x \in P, Python \Rightarrow \neg Correct(x)$

   (b) Every incorrect program is written in Python
   $\forall x \in P, \neg Correct(x) \Rightarrow Python(x)$

   (c) $\neg(\forall x \in P, Correct(x) \Rightarrow Python(x))$
   not For all programs in the set, if the program is correct, then it is written in python

   (d) $\forall x \in P, \neg Python(x) \Leftrightarrow Correct(x)$
   For all programs in the set, a program is correct if and only if it is not written in python.

2. Now let's practice translating these sattement into python expressions. First, our set-up:

   - Suppose we have a variable `programs` which stores a set of values representing Python programs. (Don't worry about the exact data type here).
   - Suppose we have a vriable `my_prog` which is one of these programs.

- Suppose we also have Python functions `is_python` and `is_correct`, which each take a "program" value and returns a `bool` representing whether that program is a "Python program" and "correct program", respectively.
  Given these Python variables and cuntsoins, wirte expressions to express each of the eight statements from Question 4 i the previsou exerciese and Question 1 in this exercise. We have done the first one for you (the others are similar but will require using and/all and comprehensions).

- Ex 1: Q4

  1. Program `my_prog` is correct and is written in Python.

```
is_correct(my_prog) and is_python(my_prog)
```

  2. At least one incorrect program is written in Python

```
any(is_correct(x) and not is_python(x) for x in programs)
```

  3. $\exists x \in P, Python(x) \wedge Correct(x)$

```
any(is_correct(x) and is_python(x) for x in programs)
```

  4. $\forall x \in P, \neg Python(x) \wedge Correct(x)$

```
all(is_correct(x) and not is_python(x) for x in programs)
```

- Ex 2: Q1

  1. No Python program is correct.

```
all(not is_python(x) or not is_correct(x) for x in programs)
```

  2. Every incorrect program is written in Python.

```
all(is_correct(x) or is_python(x) for x in programs)
```

  3. $\neg(\forall x \in P, Correct(x) \Rightarrow Python(x))$

```
not all(not is_correct(x) or is_python(x) for x in programs)
```

  4. $\forall x \in P, \neg Python(x) \Leftrightarrow Correct(x)$

```
all(not is_python(x) == is_correct(X) for x in programs)
```

3 Additional Exercises

1. So far, we have seen quantifiers only as the leftmost components of our formulas. However, because all predicate statemetns have truth values (i.e., are either True or False), they too can be combined using the standard propositional operators. Let's see some examples of this.

   (a) Using the same predicates as before, translate the following statment into English.
   $(\forall x \in P, Python(x) \Rightarrow Correct(x)) \vee (\forall y \in P, Python(y) \Rightarrow \neg Correct(y)$
   For every x in p, if a program is written in python, then it is correct or for every y in p, if a program is written in python, it is not correct.
   A program written in python is either correct or incorrect.

   (b) Again using the same predicates as before, translate the following statement into predicate logic. "If at least one python program is correct, then all python programs are correct."
   $(\exists x \in P, Python(x) \Rightarrow Correct(x)) \Rightarrow (\forall y \in P, Python(x) \Rightarrow Correct(x))$

CSC110 Lecture 6: If Statements

Hisbaan Noorani

September 23, 2020

Contents

1 Ex 1: Practice with if statements

1. Consider the code below. Note that the line are numbers on the left margin.

```
1  def can_vote(age: int) -> str:
2      """Return a string indicating whether age is a legal voting age in Canada.
3
4      In Canada, you must be at least 18 years old to vote.
5      """
6      if age < 18:
7          return 'Too young to vote'
8      else:
9          return 'Allowed to vote'
```

   (a) In the table below, write down the sequence of line numbers in the fucntions body that will be executed for the given input.

Age	Lines executed
17	1, 6, 7
18	1, 6, 9
19	1, 6, 9

   (b) Suppose we have the following doctest examples for this function.

```
1  >>> can_vote(1)
2  'Too young to vote'
3  >>> can_vote(2)
4  'Too young to vote'
```

   Why is this not a good choice of doctest examples?
   It only tests one of the branches of the if statement.

2. Implement each of the following functions using an if statement.

```
def format_name(first: str, last: str) -> str:
    """Return the first and last names as a single string in the form: last, first.

    Mononymous persons (those with no last name) should have
    their name returned without a comma.

    >>> format_name('Cherilyn', 'Sarkisian')
    'Sarkisian, Cherilyn'
    >>> format_name('Cher', '')
    'Cher'
    """
    if last == '':
        return first
    else:
        return last + ', ' + first


def larger_sum(nums1: list, nums2: list) -> list:
    """Return the list with the larger sum.

    Assume that nums1 and nums2 are lists of floats.
    In the event of a tie, return nums1.

    >>> larger_sum([1.26, 2.01, 3.30], [3.00, 3.00, 3.00])
    [3.00, 3.00, 3.00]
    >>> larger_sum([2.00, 1.00], [1.00, 2.00])
    [2.00, 1.00]
    """
    if sum(nums1) >= sum(nums2):
        return nums1
    else:
        return nums2
```

2 Ex 2: Multiple brances

1. Implement each of the following functions, using elifs to create if statements with more than two branches.

```
   def porridge_satisfaction(temperature: float) -> str:
    """Return what a picky eater says when tasting porridge with the given temperature.

    Temperatures greater than 50.0 are too hot, temperatures less than 49.0 are too cold,
    and the temperatures in between are just right.

    >>> porridge_satisfaction(65.5)
    'This porridge is too hot! Ack!!'
    >>> porridge_satisfaction(30.0)
    'This porridge is too cold! Brrr..'
    >>> porridge_satisfaction(49.5)
    'This porridge is just right! Yum!!'
    """
    if temperature > 50.0:
        return 'This porridge is too hot! Ack!!'
```

```
    elif temperature < 49.0:
        return 'This porridge is too cold! Brr..'
    else:
        return 'This porridge is just right! Yum!!'


def rock_paper_scissors(player1: str, player2: str) -> str:
    """Return the winner of a game of rock, paper, scissors.

    The game is played with the following rules:
        1) 'rock' wins against 'scissors'
        2) 'scissors' wins against 'paper'
        3) 'paper' wins against 'rock'

    Ties are allowed.

    You may assume that the input strings are in {'rock', 'paper', 'scissors'}.

    >>> rock_paper_scissors('rock', 'scissors')
    'Player1 wins'
    >>> rock_paper_scissors('rock', 'paper')
    'Player2 wins'
    >>> rock_paper_scissors('rock', 'rock')
    'Tie!'
    """
    if player1 == 'rock' and player2 == 'rock':
        return 'Tie!'
    elif player1 == 'rock' and player2 == 'paper':
        return 'Player2 wins'
    elif player1 == 'rock' and player2 == 'scissors':
        return 'Player1 wins'
    elif player1 == 'paper' and player2 == 'rock':
        return 'Player1 wins'
    elif player1 == 'paper' and player2 == 'paper':
        return 'Tie!'
    elif player1 == 'paper' and player2 == 'scissors':
        return 'Player1 wins'
    elif player1 == 'scissors' and player2 == 'rock':
        return 'Player2 wins'
    elif player1 == 'scissors' and player2 == 'paper':
        return 'Player1 wins'
    elif player1 == 'scissors' and player2 == 'scissors':
        return 'Tie!'
    else:
        return 'Error!'
```

3 Ex 3: Simplifying if statements

1. Even though this lecture is all about if statements, often we can implement predicates (functions that return booleans) without using if statements at all! For each of the following functions, rewrite the function body so that it does not use an if statement.

```python
def is_odd(n: int) -> bool:
    """Return whether n is odd (not divisible by 2).

    if n % 2 == 0:
        return False
    else:
        return True
    """
    return n % 2 != 0


def is_teenager(age: int) -> bool:
    """Return whether age is between 13 and 18 inclusive.

    HINT: identify the range of integers that make this function return True.

    if age < 13:
        return False
    else:
        if age > 18:
            return False
        else:
            return True
    """
    return age >= 13 and <= 18


def is_common_prefix(prefix: str, s1: str, s2: str) -> bool:
    """Return whether prefix is a common prefix of both s1 and s2.

    if str.startswith(s1, prefix):
        if str.startswith(s2, prefix):
            return True
        else:
            return False
    else:
        return False
    """
    return str.startswith(s1, prefix) and str.startswith(s2, prefix)


def same_corresponding_values(mapping: dict, key1: str, key2: str) -> bool:
    """Return whether the two given keys have the same corresponding value in mapping.

    Return False if at least one of the keys is not in the mapping.

    if key1 not in mapping:
        return False
    elif key2 not in mapping:
        return False
    elif mapping[key1] == mapping[key2]:
        return True
```

```
    else:
        return False
    """
```

4 Ex 4: The order of if conditions

Consider the following two functions:

```
def pick_animal1(number: int) -> str:
    """Return an animal (based on a number range)."""
    if number > 1:
        return 'Cat'
    elif number > 10:
        return 'Dog'
    else:
        return 'Duck'


def pick_animal2(number: int) -> str:
    """Return an animal (based on a number range)."""
    if number > 10:
        return 'Cat'
    elif number > 1:
        return 'Dog'
    else:
        return 'Duck'
```

We will now use our knowelede of comprehenstions, `range`, and, `all` to write expressions that descirbe the return values these two functoins.

1. Fill in the blanks in each of the following code snippets to satisfy the given description.

   (a) A set of 3 integers where `pick_animal1` only returns `'Duck'`:

   ```
   >>> all([pick_animal1(x) == 'Duck' for x in {-1, -2, -3}])
   True
   ```

   (b) A range of at least 3 integers wehre `pick_animal1` only returns `'Cat'`:

   ```
   >>> all([pick_animal1(x) == 'Duck' for x in {2, 3, 4}])
   True
   ```

   (c) A set of 3 integers where `pick_animal2` only returns `'Duck'`:

   ```
   >>> all([pick_animal2(x) == 'Duck' for x in {-1, -2, -3}])
   True
   ```

   (d) A range of at least 6 integers where `pick_animal2` only returns `'Cat'`:

   ```
   >>> all([pick_animal2(x) == 'Cat' for x in range(11, 20)])
   True
   ```

   (e) The largest range of integers where `pick_animal2` only returns `'Dog'`:

```
>>> all([pick_animal2(x) == 'Dog' for x in range(2, 10)])
True
```

2. Can `pick_animal1` ever return `'Dog'`? Why or why not

   No. If a number is greater than 10, then it is also greater than one. When the first branch of the if statment gets triggered, the interpreter ignores the rest of the branches in that block.

CSC110 Lecture 7: Function Specification and Working with Definitions

Hisbaan Noorani

September 28, 2020

Contents

1 Ex 1: Reviewing functions correctenss and writing preconditions

1. What is the relationship between a function's paramater type annotations and a function's preconditions?

   The paramater type annotations are one of the functions preconditions. The paramaters must have the specified type.

2. The following function calculates the pay for an employee who worked for a given time period (e.g., 10am–4pm) at a given hourly pay rate (e.g., \$15/hour). Write Python expressions for preconditions to express the constraints on the function inputs described in the docstring.

```
def calculate_pay(start: int, end: int, pay_rate: float) -> float:
    """Return the pay of an employee who worked for the given time at the given pay rate.

    start and end represent the hour (from 0 to 23 inclusive) that the employee
    started and ended their work.

    pay_rate is the hourly pay rate, and must be >= 15.0 (the minimum wage).

    Preconditions:
      - pay_rate >= 15.0
      - 0 <= start <= 23
      - 0 <= end <= 23
      - start <= end

    >>> calculate_pay(3, 5, 15.5)
    31.0
    >>> calculate_pay(9, 21, 22.0)
    264.0
    """
    return (end - start) * pay
```

3. Implement the function below.

```python
def ticket_price(age: int) -> float:
    """Return the ticket price for a person who is age years old.

    Seniors 65 and over pay 4.75, kids 12 and under pay 4.25, and
    everyone else pays 7.50.

    Precondition:
      - age > 0

    >>> ticket_price(7)
    4.25
    >>> ticket_price(21)
    7.5
    >>> ticket_price(101)
    4.75
    """
    elif age <= 12:
        return 4.25
    elif age >= 65:
        return 4.75
    else:
        return 7.50
```

2 Ex 2: `typing` type annotations

1. For each of the following python literal values, write down the approrpriate type annotation (from `typing`) for that value.

Python Value	Type annotation
`[1, 2, 3]`	List[int]
`{'hi', 'bye', 'haha'}`	Set[str]
`{1.5: True, 3.6: False, -1.0: True}`	Dict[float, str]
`(1, 'Hi')`	Tuple[int, str]
`([1, 2, 3], [4, 5, 6])`	Tuple[List[int], List[int]]

2. For each of the following pieces of (collections) data, write the appropriate type annotation using the `typing` module to represent that data.

Description of Data	Type Annotation
A study music playlist (song names)	Set[str]
A colour in the RGB24 model	Tuple[int, int, int]
David's grocery list (food names and quantities)	Dict[str, int]
An unordered collection of distinct points in the Cartesian plane	Set[Tuple[int, int]]

3. Why would we type the annotation `list` instead of `List[...]` (with a type in the square brackets)?

   A list may contain multiple element types so we can't descirbe all of them with `List[...]`

3 Ex 3:

1. Consider the following statement:

If m and n are odd integers, then mn is an odd integer.

If we want to express this statement using predicate logic, we need to start with a definition of the term "odd". Let $n \in \mathbb{Z}$. We say that n is odd when $2|(n-1)$. That is, n is odd when $\exists k \in \mathbb{Z}, n = 2k+1$

(a) Write the definiton of a predicate over the integers named *Odd* that is true when its argument is odd.
$Odd : \exists k \in \mathbb{Z}, n = 2k+1$, where $n \in \mathbb{Z}$
or
$Odd : 2|(n-1)$, where $n \in \mathbb{Z}$

(b) Using the predicate *Odd* and the notation of predicate logic, express the statement:

For every pair of odd integers, m and n, mn is an odd integer.

$\forall m, n \in \mathbb{Z}, (Odd(m) \wedge Odd(n)) \Rightarrow Odd(mn)$

(c) Repeat part (b) but do not use the predicates *Odd* or $|$. Instead use the full definition of odd that includes a quantified variable.
$\forall m, n \in \mathbb{Z}, \exists k_1, k_2, k_3 \in \mathbb{Z}$ s.t. $(m = 2k_1 + 1 \wedge n = 2k_2 + 1) \Rightarrow mn = 2k + 1$
or
$\forall m, n \in \mathbb{Z}, ((\exists k_1 \in \mathbb{Z}$ s.t. $m = 2k_1 + 1) \wedge (\exists k_2 \in \mathbb{Z}$ s.t. $n = 2k_2 + 1)) \Rightarrow (\exists k_3 \in \mathbb{Z}$ s.t. $mn = 2k_3 + 1)$

2. consider the following Python functions.

```
1  def average(nums: Set[int]) -> float:
2      """Return the average of a set of numbers. (Preconditions omitted)"""
3      return sum(nums) / len(nums)
4
5
6  def larger_average(nums1: Set[int], nums2: Set[int]) -> Set[int]:
7      """Return the set of numbers with the larger average. (Preconditions omitted)"""
8      if average(nums1) >= average(nums2):
9          return nums1
10     else:
11         return nums2
```

Rewrite `larger_average` so that it does not call `average`, but instead does the same calculation as the body of average directly.

(This is the equivalent of "expanding" the definition of a function.)

```
1  def larger_average(nums1: Set[int], nums2: Set[int]) -> Set[int]:
2      """Return the set of numbers with the larger average. (Preconditions ommitted)"""
3      if sum(nums1) / len(nums1) >= sum(nums2) / len(nums2):
4          return nums1
5      else:
6          return nums2
```

4 Additional Exercises

1. Repeat Exercise 3 questions 3b and 3c using the following statement (which states the converse of the original implication from that question).

For every pair of integers m and n, mn is odd, then m and n are odd.

2. Now consider a similar computational task.

(a) Define a Python function that takes an `int` and erturns whether it is odd.

(b) Define a Python function that takes a set of ~int~s, and returns a new set containing the elements of the input set that are odd.

Implement this function in two ways: using the function you defined in part (a), and not using that funciton (instead writing its body in your function for this part).

CSC Lecture 8: Porperty-Based Testing and Nested Quantifiers

Hisbaan Noorani

September 29, 2020

Contents

1 Ex 1: Property-basted testing

1. Consider the `divides` function from lecture.

```python
def divides(d: int, n: int) -> bool:
    """Return whether d divides n."""
    possible_divisors = range(- abs(n), abs(n) + 1)
    return any({n == k * d for k in possible_divisors})
```

There are many different properties of divisibility from mathematics that we can use to express property-based tests. For each of the properties below, translate them into a property-based test. We've started the first one for you (you can copy-and-paste the template and use it for each one; you don't need to repeat the import statements). You don't need to include doctests.

(a) I$\forall a, d \in \mathbb{Z}, d|ad$

```python
from hypothesis import given
from hypothesis.strategies import integers


@given(a=integers(), d=integers())
def test_a(a: int, d: int) -> None:
    assert divides(d, d * a)
```

(b) $\forall n \in \mathbb{Z}, 1|n$

```python
from hypothesis import given
from hypothesis.strategies import integers


@given(n=integers())
def test_a(n: int) -> None:
    assert divides(1, n)
```

(c) $\forall n \in \mathbb{Z}, n|n$

```python
from hypothesis import given
from hypothesis.strategies import integers


@given(n=integers())
def test_a(n: int) -> None:
    assert divides(n, n)
```

(d) $\forall d, n \in \mathbb{Z}, d|n \Rightarrow d|n+d$

```python
from hypothesis import given
from hypothesis.strategies import integers


@given(d=integers(), n=integers())
def test_a(d: int, n: int) -> None:
    assert not divides(d, n) or divides(d, n + d)
```

2 Ex 2: Nested Quantifiers

Consider the following table that shows the employees of a (very small) company:

Employee	Salary	Department
Aizah	70,000	Sales
Betty	25,000	Sales
Carlos	50,000	HR
Doug	40,000	Sales
Ellen	60,000	Design
Flo	30,000	Design

Let E be the set of six employees listed above. We'll define two predicates on this set:

$$Rich(x) : x \text{ earns more than 45,000, where } x \in E$$
$$SameDept(x, y) : x \text{ and } y \text{ are in the same department, where } x, y \in E$$

1. Consider the statement

$$\exists x, y \in E, Rich(x) \wedge SameDept(x, y)$$

Give one example to show that this statement is True. Is there more than one possible answer? Note: just as in programming, the two variables x and y can have the samve value (i.e. refer to the same employee).

Let x = Aizah.

Let y = Betty.

Aizah is rich.

They are both in sales.

Yes, there is more than one possible answer.

2. Here's the same statement, but with a universal quantifier instead:

$$\forall x, y \in E, Rich(x) \wedge SameDept(x, y)$$

Give one counterexample to show that this statement is False. Is there more than one possible answer?

Let x = Flo.

Let y = Ellen.

Flo is not rich.

Ellen and flow are in the same department but that does not matter because Flo is not rich.

Yes, there is more than one possible answer.

3. Now let's look at alternating quantifiers.

$$\forall y \in E, \exists x \in E, Rich(x) \wedge SameDept(x, y)$$

Is this True or False? How do you know?

True, there at least one rich employee in every department.

4. With the quantifiers switched:

$$\exists x \in E, \forall y \in E, Rich(x) \wedge SameDept(x, y)$$

Explain why this statement is False.

This statment claims that all people in the set E are in the same department. This is not true.

3 Ex 3: Nested quantifiers in Python

Suppose we have the following definitions in Python:

- A variable `employees` referring to a set of values representing employees (Don't worry about the exact data type here.)
- A function `is_rich` that takes an employee and returns a `bool` representing whether that employee earns more than 45,000.
- A functoin `same_dept` that takes two employees and returns whether they are in the same department.
- Given these Python variables and functions, write expressions to express each of the four statements from the previous exercise.

  Note that for the alternating quantifiers, you'll need to use nested `and/all` calls. If you have time, try writing a separate function for the inner `any/all` call to make your code easier to understand.

```
EMPLOYEES = [
    ('Aizah', 70000, 'Sales'),
    ('Betty', 25000, 'Sales'),
    ('Carlos', 50000, 'HR'),
    ('Doug', 40000, 'Sales'),
    ('Ellen', 60000, 'Design'),
    ('Flo', 30000, 'Design')
]

def is_rich(employee: tuple) -> bool:
    return employee[1] > 45000

def same_dept(employee1: tuple, employee2: tuple) -> bool:
```

```
    return employee1[2] == employee2[2]

one = any({is_rich(x) and same_dept(x, y) for x in EMPLOYEES for y in EMPLOYEES})

two = all({is_rich(x) and same_dept(x, y) for x in EMPLOYEES for y in EMPLOYEES})

three = all({any({is_rich(x) and same_dept(x, y) for x in EMPLOYEES} for y in EMPLOYEES)})

four = any({all({is_rich(x) and same_dept(x, y) for x in EMPLOYEES} for y in EMPLOYEES)})
```

CSC110 Lecture 9: Nested Data

Hisbaan Noorani

September 30, 2020

Contents

1 Ex 1: Working with nested lists

1. Here is the sample list data we saw in lecture.

```
>>> marriage_data = [
    [1657, 'ET', 80, datetime.date(2011, 1, 1)],
    [1658, 'NY', 136, datetime.date(2011, 1, 1)],
    [1659, 'SC', 159, datetime.date(2011, 1, 1)],
    [1660, 'TO', 367, datetime.date(2011, 1, 1)],
    [1661, 'ET', 109, datetime.date(2011, 2, 1)],
    [1662, 'NY', 150, datetime.date(2011, 2, 1)],
    [1663, 'SC', 154, datetime.date(2011, 2, 1)],
    [1664, 'TO', 383, datetime.date(2011, 2, 1)]
]
```

For each of the following expressions, write down what it would evaluate to.

```
>>> len(marriage_data)
8

>>> marriage_data[5]
[1662, 'NY', 150, datetime.date(2011, 2, 1)]

>>> marriage_data[0][0]
1657

>>> marriage_data[6][2]
154

>>> len(marriage_data[3])
4
```

```
>>> max([row[2] for row in marriage_data])
383
```

2. Using the same data from the previous question, write the python expressions for each of the following descriptions:

```
    >>> # Number of marriages in 'ET' in February 2011 (use list indexing to access the right row)
    >>> marriage_data[4][2]
    109

    >>> # The date corresponding to ID 1662 (use list indexing to access the right row)
    >>> marriage_data[5][3]
    datetime.date(2011, 2, 1)

    >>> # The minimum number of marriage licenses given in a month
    >>> min([row[2] for ro in marriage_data])
    80

    >>> # The rows for February 2011---use a list comprehension with a filter
    >>> [row for row in marriage_data if row[3].year == 2011 and row[3].month == 2]
    [[1661, 'ET', 109, datetime.date(2011, 2, 1)], [1662, 'NY', 150, datetime.date(2011, 2, 1)], [16
```

2 Ex 2: Constraining the "marriage license" data representation

Suppose we have a variable `marriage_data` that is a nested list representing the marriage license data we've been working with in lecture. We have listed below several statements that are constraints on the rows in this list. Your task is to translate each of these constraints into an equivalent Python expression (similar to how you translated English preconditions into Python on this week's prep).

Use the variable `marriage_data` to refer to the nested list containing all rows; each of the constraints except the last can be expressed as an expression of the form `all({____ for row in marriage_data})`.

1. Every row has length 4.

```
all({len(row) == 4 for row in marriage_data})
```

2. Ever row's first element (representing the "row id") is an integer greater than 0. (Hint: use `isinstance(___, int)` to return whether a value is an int).

```
all({row[0] > 0 and isinstance(row[0], int) for row in marriage_data})
```

3. Every row's second element (representing the civic centre) is one of `'TO'`, `'ET'`, `'NY'`, or `'SC'`.

```
all({row[1] in {'TO', 'ET', 'NY', 'SC'} for row in marriage_data})
```

4. Ever row's third element (representing the number of marriage licenses) is an integer greater than or equal to 0.

```
all({row[2] > 0 and isinstance(row[2], int) for row in marriage_data})
```

5. Every row's fourth element is a `datetime.date` value. (You ca nuse `isinstance` here as well).

```
all({isinstance(row[3], datetime.date) for row in marriage_data})
```

6. At least one row has `'TO'` as its civic centre.

```
any({row[1] == 'TO' for row in marriage_data})
```

3 Ex 3: Implementing functions on nested lists

Your task here is to implement the following functions that operate on our arriage license data set.

```
def civic_centre_dates(data: List[list]) -> Set[datetime.date]:
    """Return a set of all the dates found in data.

    Preconditions:
    - data satisfies all of the properties described in Exercise 2
    """
    return {row[3] for row in data}
```

```
def civic_centre_meets_threshold(data: List[list], civic_centre: str, num: int)\
        -> bool:
    """Return whether civic_centre issued at least num marriage licences every month.

    You only need to worry about the rows that appear in data; don't worry about "missing" months.

    Preconditions:
    - num > 0
    - data satisfies all of the properties described in Exercise 2
    - civic_centre in {'TO', 'NY', 'ET', 'SC'}

    HINT: you'll need to use a list comprehension with a filter.
    """
    filtered_data = [row for row in data if row[1] == civic_centre]
    return all({row[2] >= num for row in filtered_data})
```

```
def summarize_licences_by_centre(data: List[list]) -> Dict[str, int]:
    """Return the total number of licences issued by each civic centre in
    <data>.

    Returns a dictionary where keys are civic centre names and values are the
    total number of licences issued by that civic centre.

    Preconditions:
    - data satisfies all of the properties described in Exercise 2

    HINT: you will find it useful to write a function that calculates the total
    number of licences issued for a given civic centre as a parameter,
    e.g. total_licenses_for_centre(data, civic_centre).
    """
    return {'TO': calculate_total_marriages(data, 'TO'),
```

```
            'NY': calculate_total_marriages(data, 'NY'),
            'ET': calculate_total_marriages(data, 'ET'),
            'SC': calculate_total_marriages(data, 'SC')}

def calculate_total_marriages(data: list, civic_centre: str) -> int:
    """Return the total marriages licensed at a given civic centre"""
    return sum([row[2] for row in data if row[1] == civic_centre])
```

4 Ex 4: Investigating `datetime.date` data type

Suppose we want to find the number of marriage licenses that were issued in a particular month. To start, we would like to filter the rows of data by their month, much like we did filtering by civic centre in lecture. But the fourth element of each row is a date value; from this value, how do we check its month? In Python, a date value is composed of three pieces of data, a year, month, and day, and each of these are called attributes of the value. We can access these attributes individually using dot notation once again:

```
>>> import datetime
>>> canada_day = datetime.date(1867, 7, 1)
>>> canada_day.year  # The year attribute
1867
>>> canada_day.month  # The month attribute
7
>>> canada_day.day  # The day attribute
1
```

With this information in mind, implement the following functions:

```
def issued_licences_by_year(data: List[list], year: int) -> int:
    """Return the total number of marriage licences issued in <year>.

    Preconditions:
    - data satisfies all of the properties described in Exercise 2
    """
    return sum([row[2] for row in data if row[3].year == year])
```

```
def only_first_days(data: List[list]) -> bool:
    """Return whether every row's fourth element is a datetime.date whose <day> attribute is 1.

    Preconditions:
    - data satisfies all of the properties described in Exercise 2
    """
    return all({row[4].day == 1 for row in data})
```

```
def issued_licenses_in_range(data: List[list], start: datetime.date, end: datetime.date) -> int:
    """Return the number of marriage licenses issued between start and end,
    inclusive.

    Preconditions:
    - data satisfies all of the properties described in Exercise 2
    - end > start

```

```
    HINT: You can use <, <=, >, and >= to compare date values chronologically.
    """
    return sum([row[2] for row in data if start <= row[3] <= end])
```

5 Additional exercises

1. Express the following constraint on `marriage_data`: Every row has a unique ID (the first element).

CSC110 Lecture 10: Data Classes

Hisbaan Noorani

October 05, 2020

Contents

1 Ex 1: Reviewing data classes

1. Each code snippet below attempts to define and/or use a data class to represent a food container. Unfortunately, each has some kind of problem (syntax, logical, style, etc.) Underneath each one, identify the problem.

   Assume all the necessary imports are included—this is not the error.

   (a) Number one

```
   dataclass FoodContainer:
       """A container that can store different foods."""
       label: The name of this container
       contents: The contents of this container
```

   Problem:

   - The header should be `class` not `dataclass`.
   - No `@dataclass` header.
   - English description of variables instead of Python.

   (b) Number two

```
@dataclass
   class food_container:
       """A container that can store different foods."""
       label: str
       contents: List[str
```

   Problem:

   - `@dataclass` should use `CamelCase` and not `snake_case`

   (c) Number three

```
class FoodContainer:
        """A container that can store different foods."""
        label
        content
```

Problem:

- No `@dataclass` header.
- No description of variables.

(d) Number four

```
@dataclass
    class FoodContainer:
        """A container that can store different foods."""
        label: str
        contents: List[str]


    # In Python console
    >>> mc = FoodContainer('Nothing in here...')
```

Problem:

- You should pass an empty list instead of nothing when initializing a dataclass.
- The thing itself looks good.

2. Suppose we have the following data class definition:

```
from dataclasses import dataclass
from typing import List

@dataclass
class FoodContainer:
    """A container that can store different foods."""
    label: str
    contents: List[str]
```

Write an expression to represent a food container called `'Mario lunch box'` containing items `'sushi'` and `'chocolate'`.

```
lunch-box = FoodContainer('Mario\'s lunch box', ['sushi', 'chocolate'])
```

3. Assume we have the same data class definition as in the previous question, and we instantiate the following object:

```
>>> mc = FoodContainer('Secret snacks', ['mystery A', 'mystery B', 'mystery C'])
```

(a) Write an expression that accesses the second item in the container's contents:

```
>>> mc.contents[1]
'mystery B'
```

(b) Write an expression that would cause the following error: `AttributeError: 'FoodContainer' object has no attribute 'Contents'`.

```
>>> mc.Contents
```

(c) What does part (b) imply about writing names for accessing attributes?
They are case sensitive, and so are all things in Python.

2 Ex 2: Representation invariants

1. We have defined following data class to represent a student at the University of Toronto. Review the attributes of this data class, and then brainstorm some representation invariants for this data class. Write each one as a Python expression (using `self.<attribute>` to refer to instance attributes), and then for practice write English translations for each one.

```
@dataclass
class Student:
    """A student at the University of Toronto.

    Representation Invariants:
     - self.given_name != ''
     - self.family_name != ''
     - self.year_of_study >= 1
     - len(self.utorid) == 8
    """
    given_name: str
    family_name: str
    year_of_study: int
    utorid: str
```

2. The following data class represents data for the Computer Science Student Union. Again, review the attributes (we've also provided descriptions in the class docstring), and then translate the following representation invariants into Python expressions:

   - every clothing item's price is ≥ 0
   - every executive role is a non-empty string containing only alphabetic letters (use `str.isalpha` to check for this)
   - no student can hold more than one executive role (you can use $==$ to compare Students)

```
@dataclass
class Cssu:
    """The Computer Science Student Union at the University of Toronto.

    Instance Attributes:
        - execs: A mapping from executive role (president, treasurer, etc.)
                 to Student.
        - merch: A mapping from clothing item (t-shirt, hoodie, etc.)
                 to price.

    Representation Invariants:
     - all({self.merch[item] >= 0 for item in self.merch})
     - all({str.isalpha(role) for role in self.execs})

     - len({self.execs}) == len({self.execs[role] for role in self.execs})

    OR

     - len({(role1, role2) for role1 in self.execs for role 2 in self.execs if role1 != role2 and se

    OR
```

```
    - not any({self.execs[role1] == self.exec[role2] for role1 in self.execs for role2 in self.exec
    """
    execs: Dict[str, Student]
    merch: Dict[str, float]
```

3 Ex 3: Marriage Licences revisited

In our last lecture we used a nested list to represent a table of marriage license data:

ID	Civic Centre	Marriage Licenses Issued	Time Period
1657	ET	80	January 2011
1658	NY	136	January 2011
1659	SC	159	January 2011
1660	TO	367	January 2011
1661	ET	109	February 2011
1662	NY	150	February 2011
1663	SC	154	February 2011
1664	TO	383	February 2011

In this lecture, we saw how to define this as a dataclass:

```
@dataclass
class MarriageData:
    """A record of the number of marriage licenses issued in a civic centre
    in a given month.

    Instance Attributes:
        - id: a unique identifier for the record
        - civic_centre: the name of the civic centre
        - num_licenses: the number of licenses issued
        - month: the month these licenses were issued

    Representation Invariants:
        - self.civic_centre in {'TO', 'ET', 'NY', 'SC'}
        - self.num_licenses >= 0
        - self.month.day == 1

    >>> row_1657 = MarriageData(1657, 'ET', 80, datetime.date(2011, 1, 1))
    """
    id: int
    civic_centre: str
    num_licenses: int
    month: datetime.date
```

In this exercise, you'll apply what you've learned about data classes to redo some of the computations from Lecture 9's worksheet using data classes instead of lists.

1. (warm-up) Using the above data class definition, write an expression to represent the row with id 1662 in the above table.

```
>>> {mc for mc in marriage_data if mc.id == 1662}
```

2. Write representation invariants for this data class to represent each of the following:

   - Civic centres must be one of 'TO', 'ET', 'NY', or 'SC'.
   - The number of marriage licenses is greater than or equal to 0.
   - The month value has a day value of 1.
     Done in the above dataclass.

3. Implement each of the following functions, which are equivalent to the ones from the previous worksheet, except they now take in a List[MarriageData] rather than a nested list.

```
def civic_centres(data: List[MarriageData]) -> Set[str]:
    """Return a set of all the civic centres found in data.
    """
    return {row.marriage_center for row in data}


def civic_centre_meets_threshold(data: List[MarriageData], civic_centre: str, num: int) -> bool:
    """Return whether civic_centre issued at least num marriage licences every
    month.

    You only need to worry about the rows that appear in data; don't worry about
    "missing" months.

    Preconditions:
        - civic_centre in {'TO', 'NY', 'ET', 'SC'}

    HINT: you'll need to use a list comprehension with a filter.
    """
    filtered_data = [row for row in data if row.civic_center = civic_center]
    return all({row.civic_center >= num for row in filtered data})


def issued_licenses_in_range(data: List[MarriageData], start: datetime.date, end: datetime.date) ->
    """Return the number of marriage licenses issued between start and end,
    inclusive.

    Preconditions:
        - end > start

    HINT: You can use <, <=, >, and >= to compare date values chronologically.
    """
    return sum([row.num_licenses for row in data if start <= row.month <= end])
```

CSC110 Lecture 11: For Loops

Hisbaan Noorani

October 06, 2020

Contents

1 Ex 1: Practice with for loops

1. Consider the following function.

```
1  def sum_of_squares(numbers: List[int]) -> int:
2      """Return the sum of the squares of the given numbers.
3
4      >>> sum_of_squares([4, -2, 1])  # 4 ** 2 + (-2) ** 2 + 1 ** 2
5      21
6      """
7      sum_so_far = 0
8
9      for number in numbers:
10         sum_so_far = sum_so_far + number ** 2
11
12     return sum_so_far
```

   (a) What is the loop variable?

   `number`

   (b) What is the accumulator?

   `sum_so_far`

   (c) Fill in the loop accumulation table for the call to function ~$\text{sum}_{\text{ofsquares}}$([4, -2, 1]).

Iteration	Loop Variable	Loop accumulator
0	-	0
1	4	16
2	-2	20
3	1	21

2. Implement the following function.

```python
def long_greeting(names: List[str]) -> str:
    """Return a greeting message that greets every person in names.

    Each greeting should have the form "Hello <name>! " (note the space at the end).
    The returned string should be the concatenation of all the greetings.

    >>> long_greeting(['David', 'Mario'])  # Note the "extra" space at the end
    'Hello David! Hello Mario! '
    """
    greeting = ''

    for name in names:
        greeting = greeting + 'Hello ' + name + '! '

    return greeting
```

2 Ex 2: Marriage license data revisisted.

In Lecture 9, we saw how to query marriage license data using a nested list (i.e., `List[list]`). In Lecture 10, we saw how to use data classes to store the marriage license data using a list of `MarriageData` (i.e., `List[MarriageData]`):

```python
@dataclass
class MarriageData:
    """..."""
    id: int
    civic_centre: str
    num_licenses: int
    month: datetime.date
```

Each of the following sets of functions takes marriage license data and performs some aggregation of those values. The first function in each set is implemented for you and uses a nested list (as we did at the end of last week). Your task is to implement each of the other two functions in the set in two ways: first with a comprehension, and second with a for loop.

1. Group 1.

```python
def total_licenses_for_centre_v1(data: List[list], civic_centre: str) -> int:
    """Return how many marriage licenses were issued in the given civic centre."""
    return sum([row[2] for row in data if row[1] == civic_centre])
```

```python
def total_licenses_for_centre_v2(data: List[MarriageData], civic_centre: str) -> int:
    """Return how many marriage licenses were issued in the given civic centre."""
    return sum([row.num_licenses for row in data if row.civic_centre == civic_centre])
```

```python
def total_licenses_for_centre_v3(data: List[MarriageData], civic_centre: str) -> int:
    """Return how many marriage licenses were issued in the given civic centre."""
    total = 0

    for row in data:
        if row.civic_centre == civic_centre:
            total = total + row.num_licenses
```

```
    return total
```

2. Group 2.

```
def civic_centre_meets_threshold_v1(data: List[list], civic_centre: str, num: int) -> bool:
    """Return whether civic_centre issued at least num marriage licences every month.

    You only need to worry about the rows that appear in data; don't worry about "missing" months.

    Preconditions:
        - num > 0
        - civic_centre in {'TO', 'NY', 'ET', 'SC'}
        - data satisfies all of the properties described in Worksheet 9, Exercise 2
    """
    licenses_issued = [row[2] for row in data if row[1] == civic_centre]
    return all(num_issued >= num for num_issued in licenses_issued)
```

```
def civic_centre_meets_threshold_v2(data: List[MarriageData], civic_centre: str, num: int) -> bool:
    """Return whether civic_centre issued at least num marriage licences every month.

    You only need to worry about the rows that appear in data; don't worry about "missing" months.

    Preconditions:
        - num > 0
        - civic_centre in {'TO', 'NY', 'ET', 'SC'}
    """
    licenses_issued = [row.num_licenses for row in data if row.civic_centre == civic_centre]
    return all(num_issued >= num for num_issued in licenses_issued)
```

```
def civic_centre_meets_threshold_v3(data: List[MarriageData], civic_centre: str, num: int) -> bool:
    """Return whether civic_centre issued at least num marriage licences every month.

    You only need to worry about the rows that appear in data; don't worry about "missing" months.

    Preconditions:
        - num > 0
        - civic_centre in {'TO', 'NY', 'ET', 'SC'}
    """
    for row in data:
        if row.civic_centre == civic_centre and row.num_licenses < num:
            return False

    return True
```

3 Ex 3: Looping over other data types

For each function, add at least one example to the docstring and complete the function body.

1. One

```python
def count_uppercase(s: str) -> int:
    """Return the number of uppercase letters in s.

    >>> count_uppercase('HELLO')
    5

    >>> cout_uppercase('Hisbaan')
    1
    """
    total_upper = 0

    for character in s:
        if str.isupper(character)
            total_upper = total_upper + 1

    return total
```

2. Two

```python
def all_fluffy(s: str) -> bool:
    """Return whether every character in s is fluffy.

    Fluffy characters are those that appear in the word 'fluffy'.

    >>> all_fluffy('fluffy')
    True

    >>> all_fluffy('fffff')
    True

    >>> all_fluffy('hello)
    False
    """
    for character in s:
        if character not in 'fluffy':
            return False

    return True
```

3. Three

```python
def sum_davids(scores: Dict[str, int]) -> int:
    """Return the sum of all values in scores that correspond to a key that contains 'David'.

    >>> I can't be bothered to type out a dictionary.
    """
    sum = 0

    for score in scores:
        if 'David' in score:
            sum = sum + scores[score]

    return sum
```

4. Four

```
def david_vs_mario(scores: Dict[str, int]) -> str:
    """Return the name of the person with the highest total score in scores.

    David's score is the sum of all values in scores that correspond
    to a key that contains the string 'David'.

    Mario's score is the sum of all values in scores that correspond
    to a key that contains the string 'Mario'.

    >>> I can't be bothered to type out a dictionary.
    """
    mario_score = 0
    david_score = 0

    for mario in scores:
        if 'Mario' in mario:
            mario_score = mario_score + scores[mario]

    for david in scores:
        if 'David' in david:
            david_score = david_score + scores[mario]

    if mario_score >= david_score:
        return 'Mario'
    else:
        return 'David'
```

4 Additional Exercises

Implement the function below in two ways: first using comprehensions, and second using a for loop.

```
def count_anomalies(data: List[MarriageData]) -> int:
    """Return the number of months where there is at least one
    civic centre differing by at least 100 from the average number
    of marriage licenses.
    """
```

CSC110 Lecture 12: More With For Loops

Hisbaan Noorani

October 07, 2020

Contents

1 Ex 1: Looping with indexes

1. Consider the following function, which we studied last class.

```
def all_fluffy(s: str) -> bool:
    """Return whether every character in s is fluffy.

    Fluffy characters are those that appear in the word 'fluffy'.

    >>> all_fluffy('fffffuy')
    True
    >>> all_fluffy('abcfluffy')
    False
    """
    for letter in s:
        if letter not in 'fluffy':
            return False

    return True
```

In the space below, rewrite the body of the function so that it uses an index-based for loop instead of the element-based for loop.

```
def all_fluffy(s: str) -> bool:
    """Return whether every character in s is fluffy.

    Fluffy characters are those that appear in the word 'fluffy'.

    >>> all_fluffy('fffffuy')
    True
    >>> all_fluffy('abcfluffy')
    False
    """
    for i in range(len(s)):
        if s[i] not in 'fluffy':
```

```
            return False

    return True
```

2. Implement each of the following functions using index-based for loops.

```
def is_sorted(lst: List[int]) -> bool:
    """Return whether lst is sorted.

    A list L is sorted when for every pair of *adjacent* elements
    x and y in L, x <= y.

    Lists of length < 2 are always sorted.

    >>> is_sorted([1, 5, 7, 100])
    True
    >>> is_sorted([1, 2, 1, 2, 1])
    False
    """
    for i in range(0, len(lst) - 1):
        if lst[i] > lst[i + 1] :
            return False

    return True
```

```
def inner_product(nums1: List[float], nums2: List[float]) -> float:
    """Return the inner product of nums1 and nums2.

    The inner product of two lists is the sum of the products of the
    corresponding elements of each list:

        sum([nums1[i] * nums2[i] for i in range(0, len(nums1))])

    Preconditions:
        - len(nums1) == len(nums2)

    >>> inner_product([1.0, 2.0, 3.0], [0.5, 2.5, 0.0])
    5.5
    """
    sum_so_far = 0

    for i in range(len(nums1)):
        sum_so_far = sum_so_far + (nums1[i] * nums2[i])

    return sum_so_far
```

```
def stretch_string(s: str, stretch_factors: List[int]) -> str:
    """Return a string consisting of the characters in s, each repeated
    a given number of times.


```

```
    Each character in s is repeated n times, where n is the int at the
    corresponding index in stretch_factors.
    For example, the first character in s is repeated stretch_factors[0] times.

    Preconditions:
        - len(s) == len(stretch_factors)
        - all({factor >= 0 for factor in stratch_factors})

    >>> stretch_string('David', [2, 4, 3, 1, 1])
    'DDaaaavvvid'
    >>> stretch_string('echo', [0, 0, 1, 5])
    'hooooo
    """
```

2 Ex 2: Nested Loops

1. Implement this function:

```python
def total_mice(dict_of_cats: Dict[str, List[str]]) -> int:
    """Return the number of mice stored in the given cat dictionary.

    dict_of_cats is a dictionary here:
        - Each key is the name of a cat
        - Each corresponding value is a list of items that the cat owns.
          An item is a *mouse* when it contains the string 'mouse'.
          (You can use the "in" operator to check whether one string is
          in another.)

    >>> total_mice({'Romeo': ['mouse 1', 'my fav mouse', 'flower'],
    ...             'Juliet': ['sock', 'mouse for tonight']})
    3
    >>> total_mice({'Asya': ['chocolate', 'toy'], 'Mitzey': []})
    0
    """
    num_of_mice = 0

    for cat_name in dict_of_cats:
        for item in range(len(dict_of_cats[cat_name])):
            if 'mouse' in item:
                num_of_mice = num_of_mice + 1

    return num_of_mice
```

2. Complete the following loop accumulation table to trace the sample function call `total_mice({'Romeo': ['mouse', 'my fav mouse', 'flower'], 'Juliet': ['sock', 'dinner mouse']})`. (We've started it for you to save some time.)

Outer Loop Iteration	Outer Loop Variable	Inner Loop Iteration	Inner Loop Variable	Accumulator
0	n/a	n/a	n/a	0
1	'Romeo'	0	n/a	0
1	'Romeo'	1	'mouse'	1
1	'Romeo	2	'my fav mouse'	2
1	'Romeo	3	'flower'	2
2	'Juliet'	0	n/a	2
2	'Juliet	1	'sock'	2
2	'Juliet	2	'dinner mouse'	3

3. Implement this function using a nested loop.

```
def can_pay_with_two_coins(denoms: Set[int], amount: int) -> bool:
    """Return whether the given amount is the sum of two distinct numbers
    from denoms.

    >>> can_pay_with_two_coins({1, 5, 10, 25}, 35)
    True
    >>> can_pay_with_two_coins({1, 5, 10, 25}, 12)
    False
    """

    # check every combination of two coins

    for i in range(len(denoms):
        for j in range(len(denoms)):
            if denoms[i] + denoms[j] and denoms[i] != denoms[j]:
                return True

    return False
```

4. Implement this function using a nested loop.

```
import math


def max_average(lists_of_numbers: List[List[float]]]) -> float:
    """Return the largest average of the given lists_of_numbers.

    Preconditions:
        - lists_of_nubers != []
        - all({numbers != [] for numbers in lists_of_numbers})

    >>> max_average([[1.0, 3.4], [3.5, 4.0, -2.5]])
    2.2
    """
    # ACCUMULATOR max_so_far: keep track of the maximum average of the lists
    # visited so far. We initialize to negative infinity so that any
    # computed average will be greater than the starting value.
    # (i.e., for all floats x, x > -math.inf)
    max_so_far = -math.inf

    for list in lists_of_nubers:
```

```
        average = sum(list) / len(liss)
        if average > max_so_far:
            max_so_far = average

    return max_so_far
```

3 Additional Exercises

1. Write a function that takes a string `s` and returns whether s is a palindrome. A palindrome is a string consists of the same sequence of characters in left-to-right order as right-to-left order. `'davad'` is a palindrome, and `'david'` is not.

2. Write a function that takes two lists of integers, which have the same length and are non-empty, and returns the greatest absolute difference between the numbers at corresponding positions in the lists.

3. Write a new version of `max_average` that does the same thing, except it returns the list with the highest average rather than the highest average.

4. Hint: use two accumulator variables, one to keep track of the highest average itself, and another to keep track of the list with the highest average.

5. Re-implement all of the functions on this worksheet using comprehensions. You might need to define some separate functions as well.

CSC Lecture 13: Variable Reassignment and Object Mutation

Hisbaan Noorani

October 13, 2020

Contents

1 Ex 1: Reassignmnet and mutation practice

1. .

   (a) Consider this code:

```
1  x = 4
2  y = 5
3  x = 2
```

   Complete the value-based memory model table to show the values of the variables after this code executes. Show both the old and new values of any variables that are reassigned.

Variable	Value
x	~~4~~ $\rightarrow$ 2
y	5

   (b) Consider this code:

```
1  x = 'hi'
2  y = x + 'bye'
3  x = y + x
```

   Complete the value-based memory model table to show the values of the variables after this code executes. Show both the old and new values of any variables that are reassigned.

Variable	Value
x	~~'hi'~~ $\rightarrow$ 'hibyehi'
y	'hibye'

2. Suppose we execute this statement in the Python console.

```
1  numbers = [1, 0]
```

   All of the following statements cause `numbers` to refer to the list value `[1, 0, 8]`. For each one, state whether the statement mutates the original list or reassigns `numbers` to a new list object.

   (a) `list.append(numbers, 8)`
   Mutates

(b) `number = numbers + [8]`
Reassigns

(c) `list.insert(numbers, 2, 8)`
Mutates

(d) `numbers = [numbers[0], numbers [1], 8]`
Reassigns

3. Suppose we execute the following code:

```
lst1 = [1, 0, 8]
lst2 = list.sort(lst1)
```

After the code above is executed, which of the following expressions evaluate to `True`? Circle those expression(s).

(lst1 == [1, 0, 8])	lst1 == [0, 1, 8]
lst2 == [1, 0, 8]	lst2 == [0, 1, 8]

4. Circle the `set` operations that mutate the input set. Try calling `help` on each function, and/or looking them up in A.2 Python Built-In Data Types Reference.

set.intersection	(set.remove)
(set.add)	set.union

5. Suppose we execute the following code:

```
animals = {'fish': {'swim'},
           'kangaroo': {'hop'},
           'frog': {'swim', 'hop'}}
```

Indicate whether each statement will cause an error and, if not, whether the statement will increase the number of key/value pairs in the dictionary:

Statement	Error? (Y/N)	Increases `len(animals)`? (Y/N)
animals['human'] = {'swim', 'run', 'walk'}	N	Y
set.add(animals['monkey'], 'swing')	Y	N
set.add(animals['kangaroo'], 'airplane')	N	N
animals['frog'] = {'tapdance'}	N	N
animals['dolphin'] = animals['fish']	N	Y

6. Read the following function's header and description, and then complete its doctests and implement the function body.

```
def move_item(items: list, other_items: set) -> None:
    """Remove the first item from items and add it to other_items.

    Preconditions:
        - items != []


    >>> numbers_list = [1, 2, 3]
```

```
    >>> numbers_set = {10, 20}
    >>> move_item(numbers_list, numbers_set)
    >>> numbers_list
    [2, 3]
    >>> numbers_set == {10, 20, 1}
    True
    """
    other_items.add(items.pop(0))
```

2 Ex 2: Loopswith collection accumulators

Recall our marriage license dataset, where we represent each row of data using the following data class:

```
@dataclass
class MarriageData:
    """A record of the number of marriage licenses issued in a civic centre
    in a given month.

    Instance Attributes:
      - id: a unique identifier for the record
      - civic_centre: the name of the civic centre
      - num_licenses: the number of licenses issued
      - month: the month these licenses were issued

    Representation Invariant omitted.
    """
    id: int
    civic_centre: str
    num_licenses: int
    month: datetime.date
```

Implement each of the following functions using a loop with an accumulator of the appropriate collection data type. Use mutating operations to avoid creating multiple collection objects.

1. .

```
def filter_by_name(data: List[MarriageData], name: str) -> List[MarriageData]:
    """Return all rows in data with the matching civic centre <name>.

    Equivalent to:
      [row for row in data if row.civic_centre == name]
    """
    rows_so_far = []

    for row in data:
        if row.civic_centre == name:
            rows_so_far.append(row)

    return rows_so_far
```

2. .

```python
def num_issued_by(data: List[MarriageData], centre: str) -> Set[int]:
    """Return the unique numbers of marriage licenses issued in a month at the
    given civic centre.

    Equivalent to:
      {row.num_licenses for row in data if row.civic_centre == name}
    """
    num_so_far = set()

    for row in data:
        if row.civic_centre == name:
            num_so_far.add(row.num_licenses)

    return num_so_far
```

3. .

```python
def marriages_by_centre(data: List[MarriageData],
                        month: datetime.date) -> Dict[str, int]:
    """Return mapping from civic centre name to the number of marriage licenses
    issued by that centre in the given month.

    Preconditions:
        - Each civic centre has only one row of MarriageData for the given month.

    Equivalent to:
      {row.civic_centre: row.num_licenses for row in data if row.month == month}
    """
    record_so_far = {}

    for row in data:
        if row.month == month:
            record_so_far[row.centre] = (row.num_licenses)

    return record_so_far
```

CSC110 Lecture 14: The Python Memory Model

Hisbaan Noorani

October 13, 2020

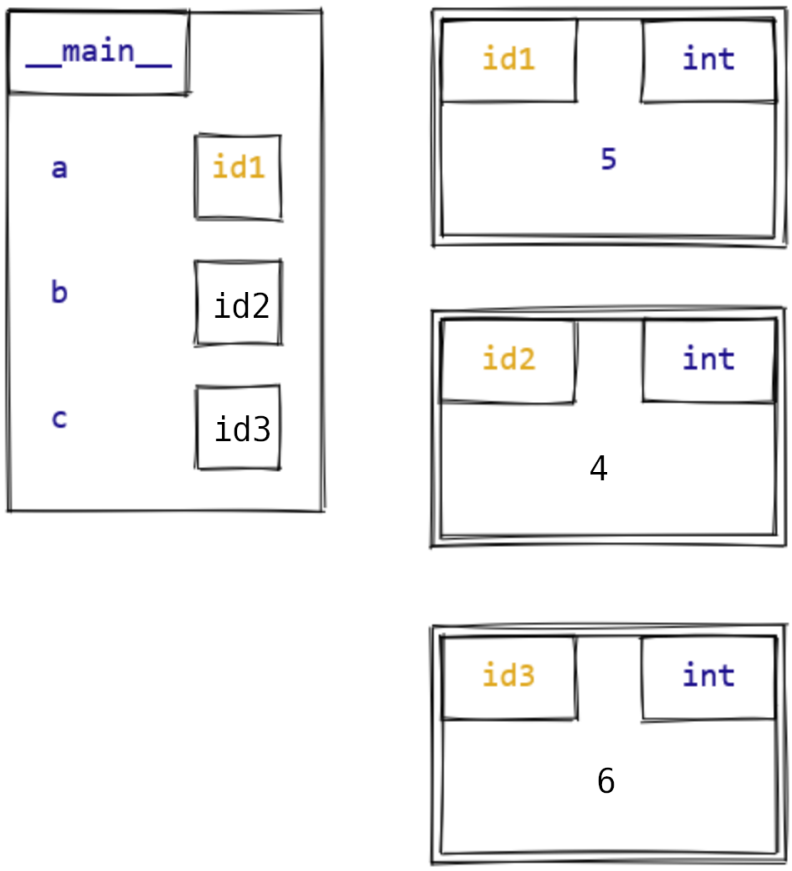
Contents

1 Exercise 1: Practice with the Python memory model

For each of the following code snippets, fill in the memory model diagram on the right to reflect the state of memory after the code is executed. In each case, we begin with the state where `a` has already been assigned the value `5`. You may not need to use all of the boxes.

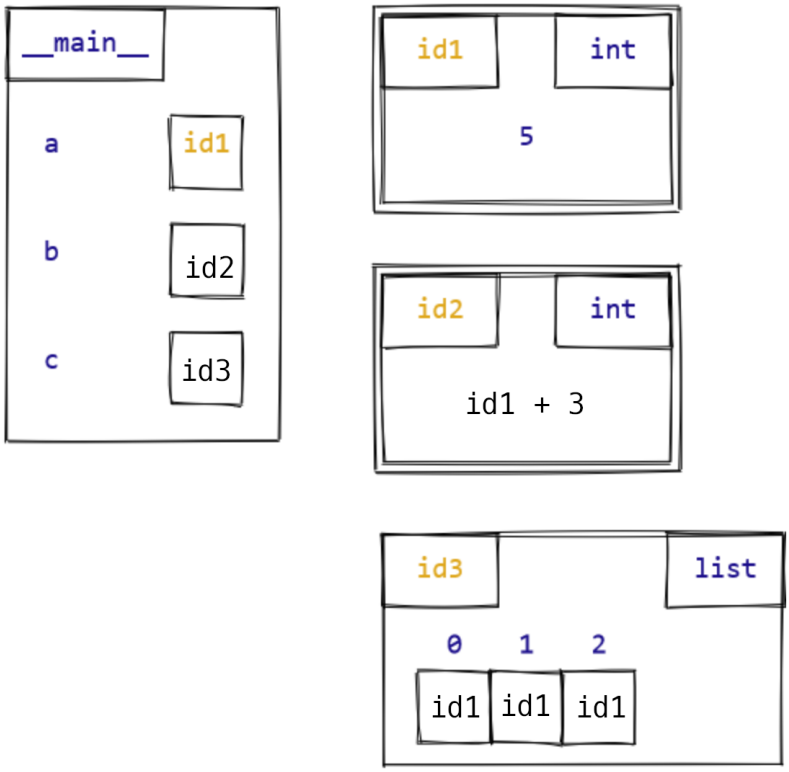
1. .

```
a = 5
b = 4
c = 6
```



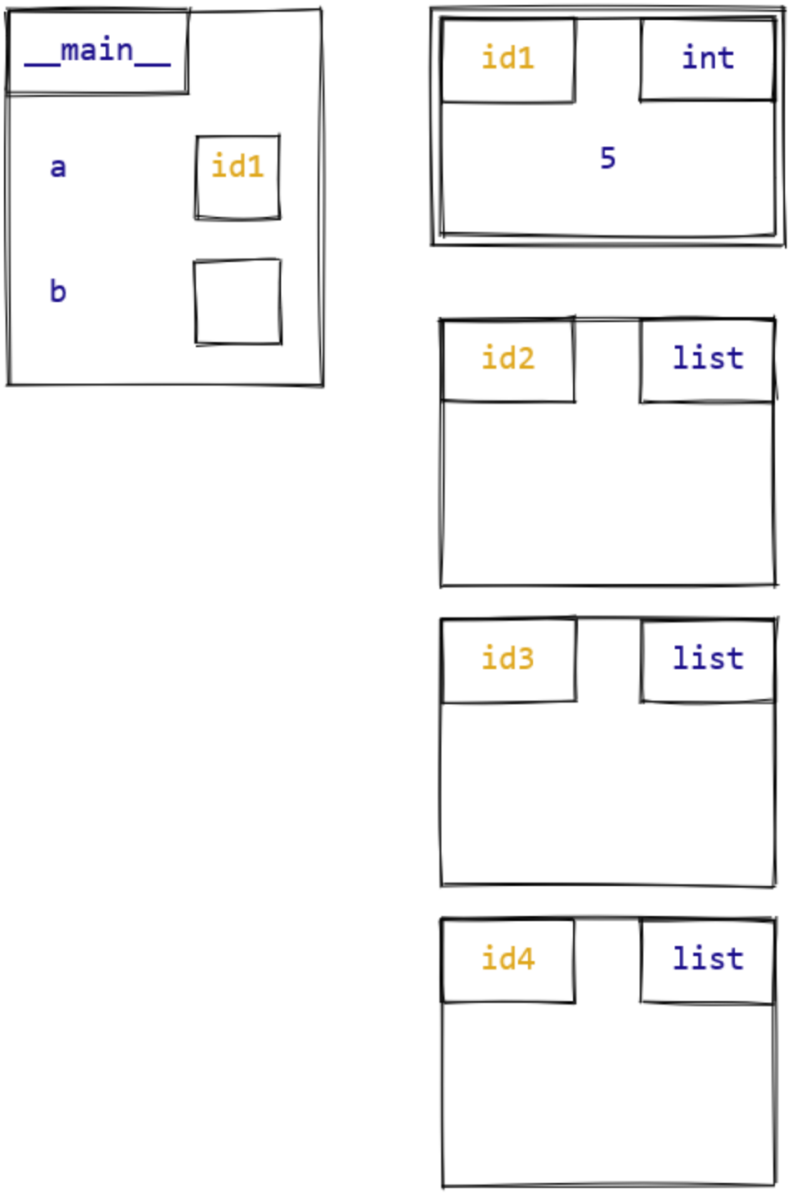
2. .

```
a = 5
b = a + 3
c = [a, a, a]
```


__main__
a
id1
b
id2
c
id3
id1
int
5
id2
int
id1 + 3
id3
list
0
1
2
id1
id1
id1

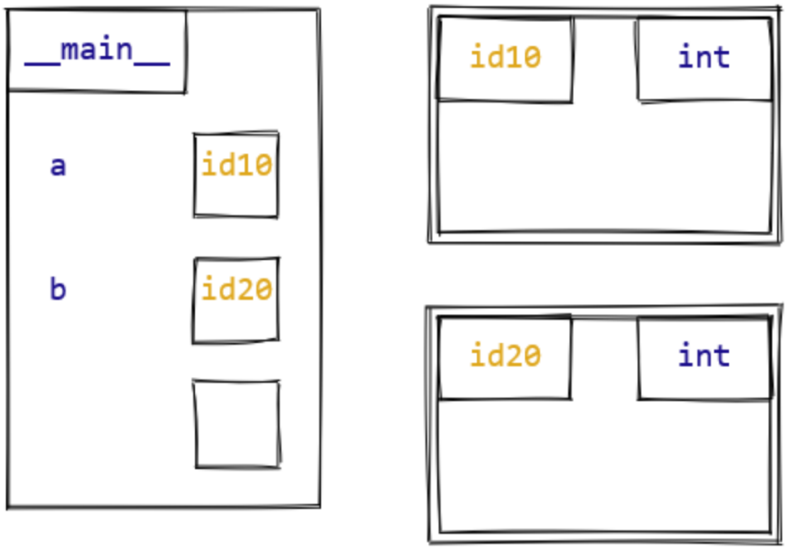
3. .

```
a = 5
b = [a]
```



2 Exercise 2: Variable reassignment and mutation with the memory model

1. Suppose we have two variables `a` and `b` that have been assigned `int` values:

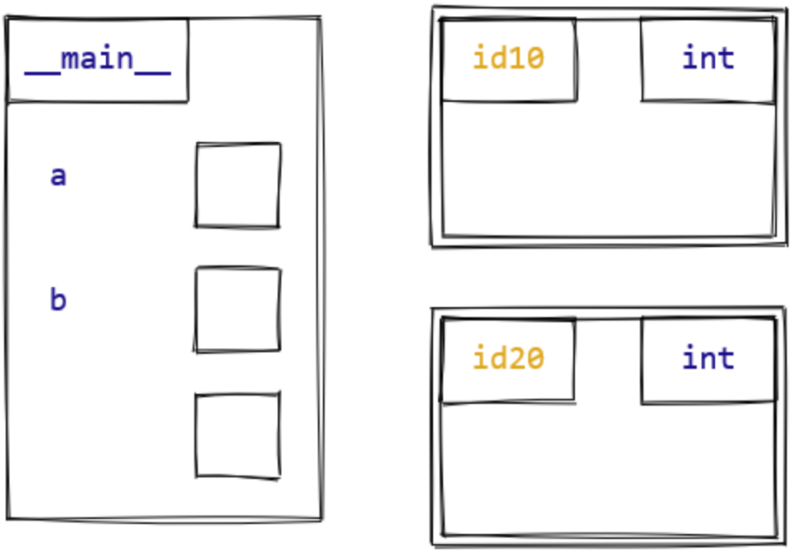


(a) Write a snippet of code to swap which values `a` and `b` refer to. After your statements are executed, `a` should refer to the object that `b` used to refer to, and `b` should refer to the object that `a` used to refer to.

Use a third "temporary" variable to perform the swap.

```
a = temp
a = b
b = temp
```

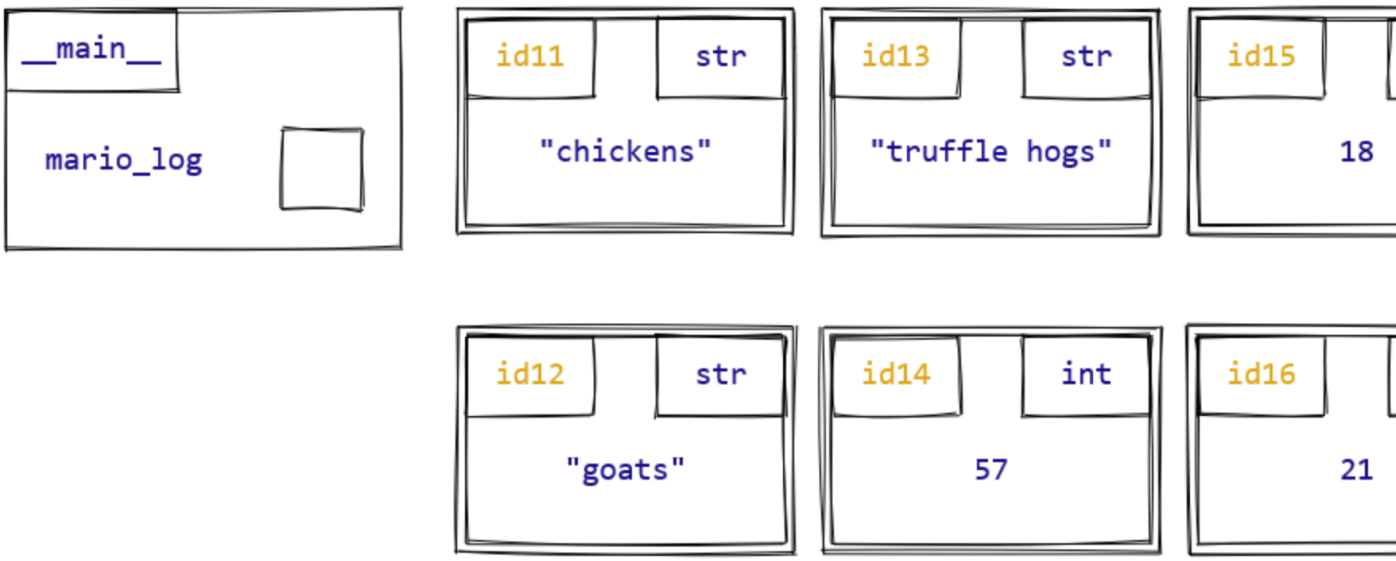
(b) Complete the following memory model diagram to show the state of memory *after* your code snippet executes.



(c) Would the code you wrote in part (a) if `a` and `b` referred to mutable values like ~list~s instead of ~int~s?

1. Farmer Mario is the proud owner of 57 chickens, 18 goats, and 21 truffle hogs. He is very organized and keeps a log of how many animals of each type he has by the end of each month.

(a) Complete the following memory model diagram to represent a variable `mario_log` to refer to this data:



(b) At the end of the month, Mario sells 5 truffle hogs to David. Write a line of code to mutate `mario_log` to reflect this change.

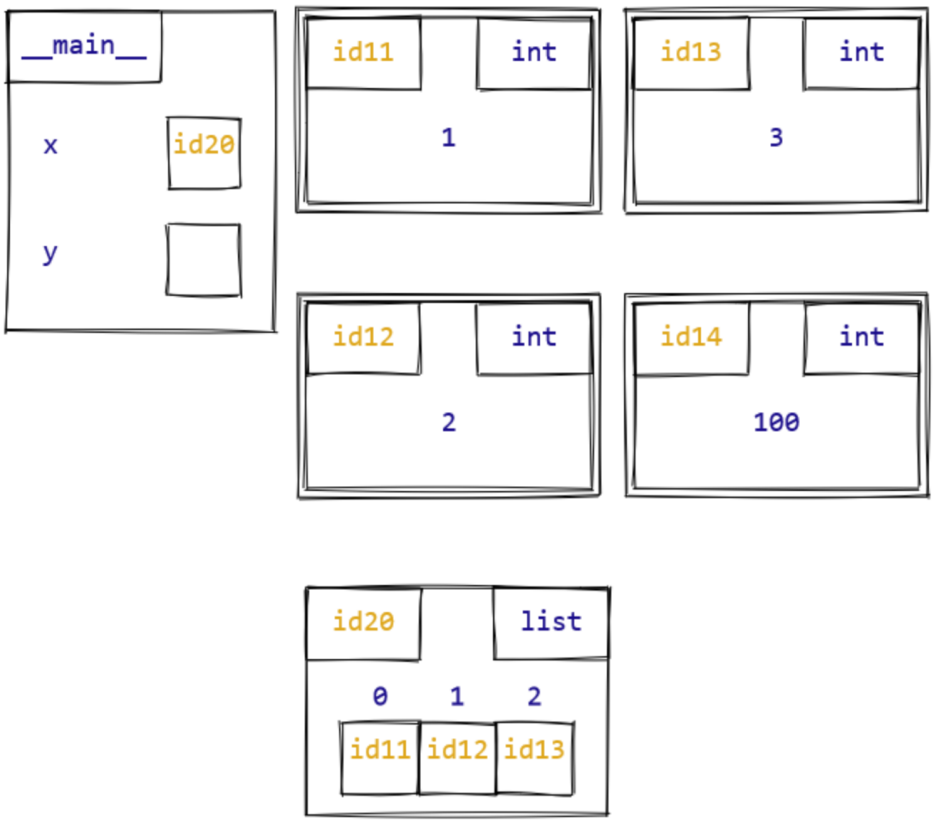
(c) Modify your above diagram to show this change.

3 Exercise 3: Aliasing

For each snippet of code below, complete the memory model diagram and answer the question below it. (You may need to indicate elements being added/removed from a collection data type.)

1. .

```
1    x = [1, 2, 3]
2    y = x
3    y = y + [4]
```



After the code above is executed, which of the following expressions evaluate to `True`? Circle those expression(s).

(a) `x == [1, 2, 3]` and `y == [1, 2, 3, 4]`

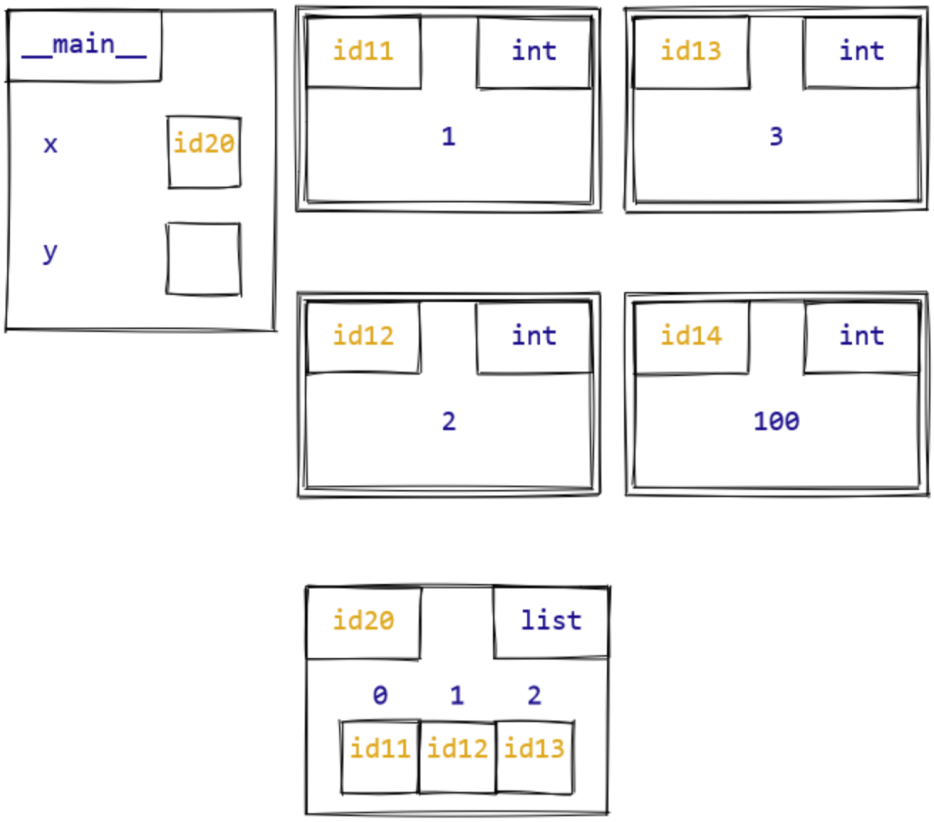
(b) `x == [1, 2, 3, 4]` and `y == [1, 2, 3, 4]`

(c) `x == [1, 2, 3]` and `y == [1, 2, 3]`

(d) `x is y`

2. Consider this code

```
1    x = {1, 2, 3}
2    y = x
3    set.add(y, 4)
```



After the code above is executed, which of the following expressions evaluate to `True`? Circle those expression(s).

(a) `x = {1, 2, 3} and y = {1, 2, 3, 4}`

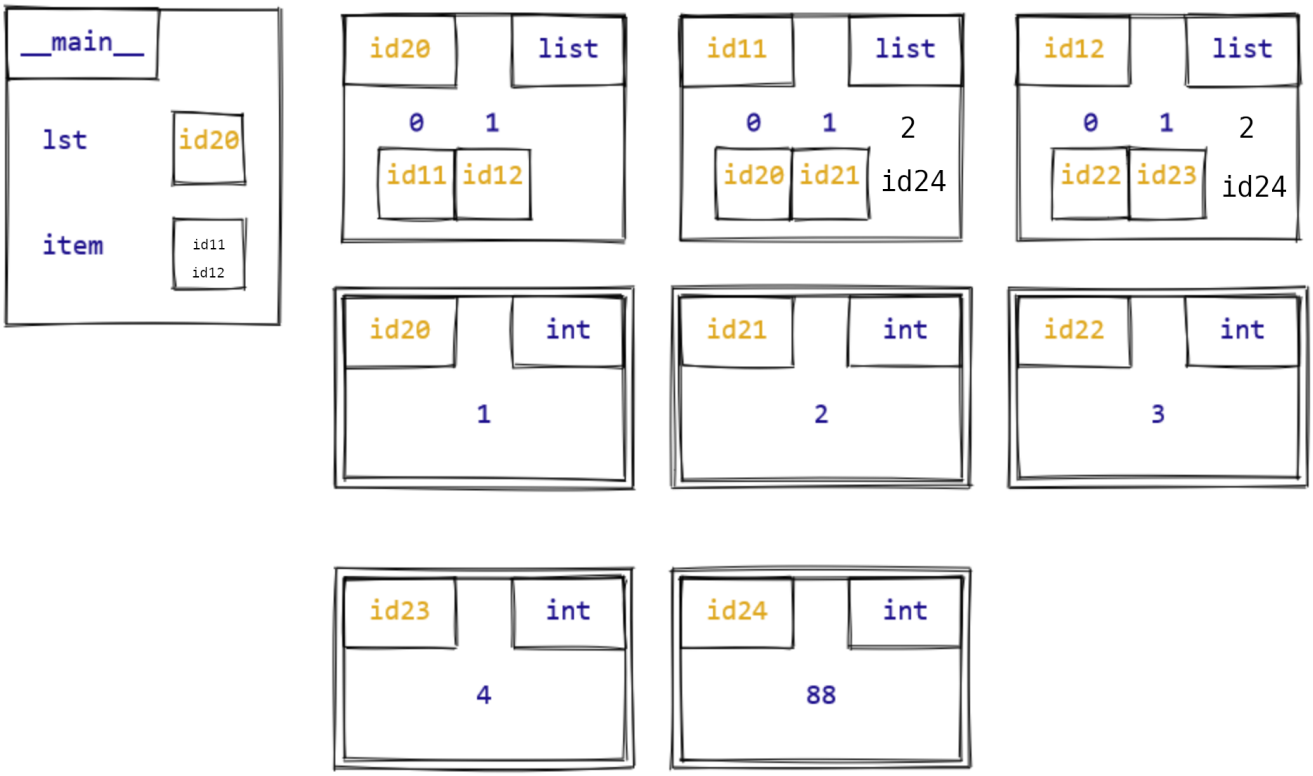
(b) `x = {1, 2, 3, 4} and y = {1, 2, 3, 4}`

(c) `x = {1, 2, 3} and y = {1, 2, 3}`

(d) `x is y`

3. .

```
lst = [[1, 2], [3, 4]]
for item in lst:
    list.append(item, 88)
```

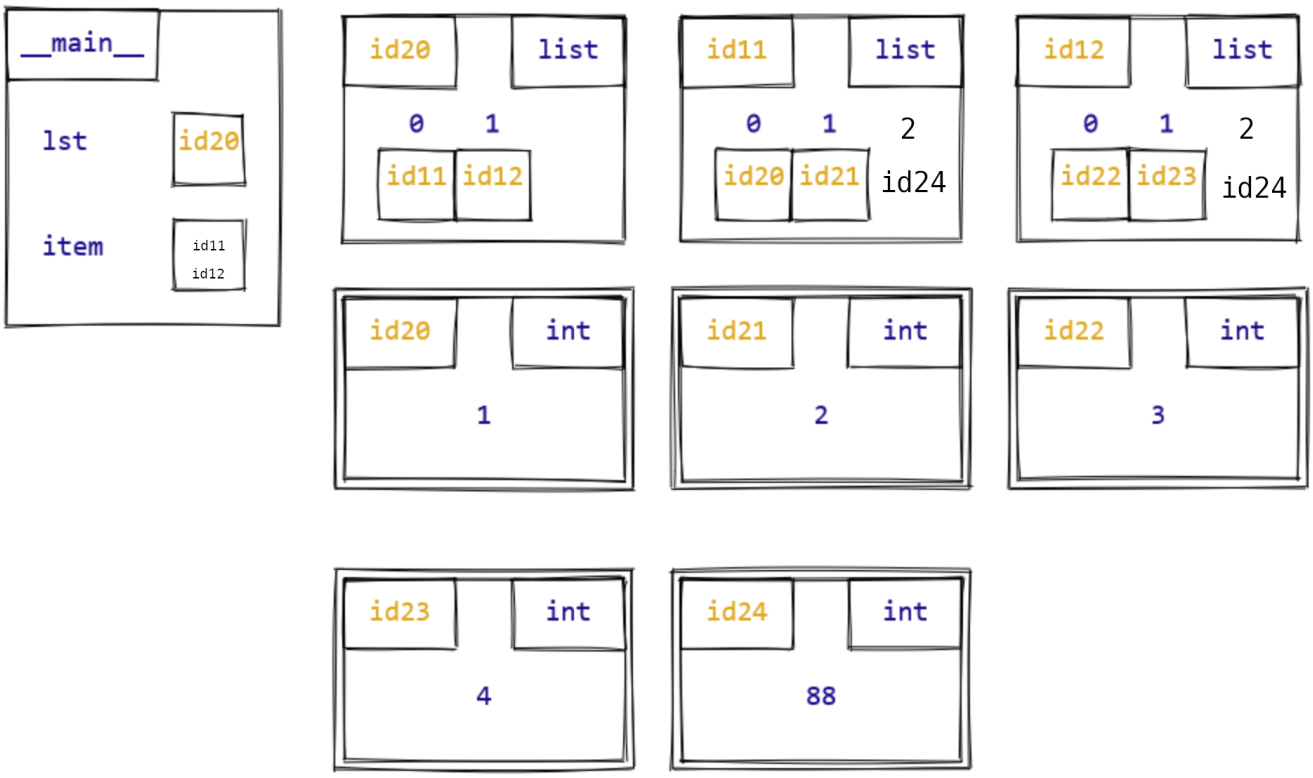


What is the value of `lst` at the end of this code snippet?

`lst = [[1, 2, 88], [1]]]`

4. .

```
lst = [[1, 2], [3, 4, 88]]
for item in lst:
    item = item + [88]
```



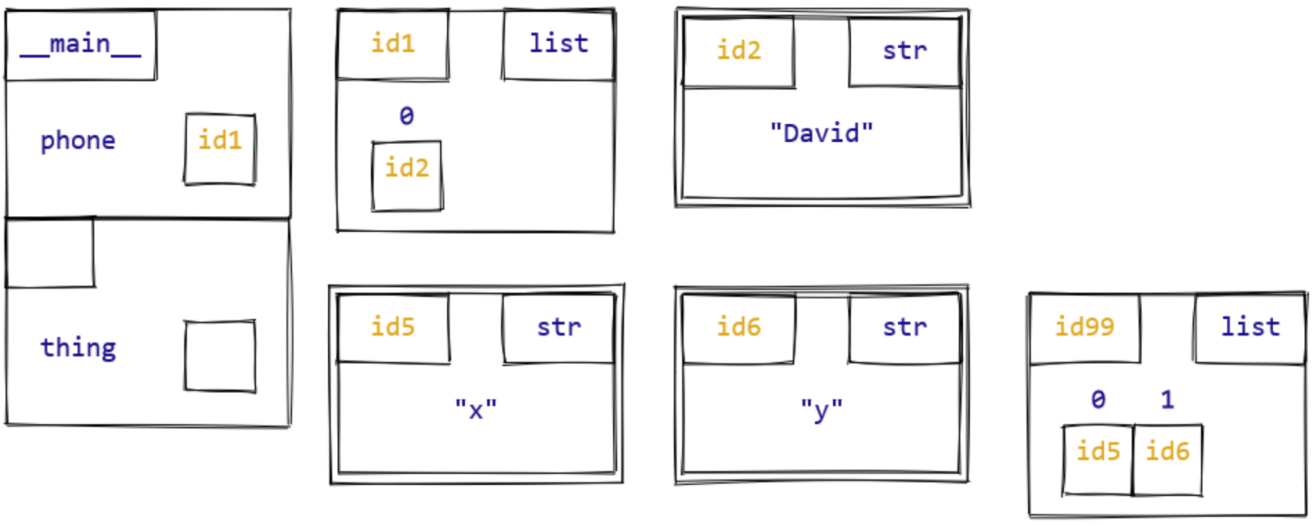
What is the value of `lst` at the end of this code snippet?

4 Exercise 4: Functions and the call stack

For each snippet of code below, draw the memory model diagram for the state of the program *immediately before* the function (`f1=/=f2`) returns. Then, write what would be displayed in the Python console.

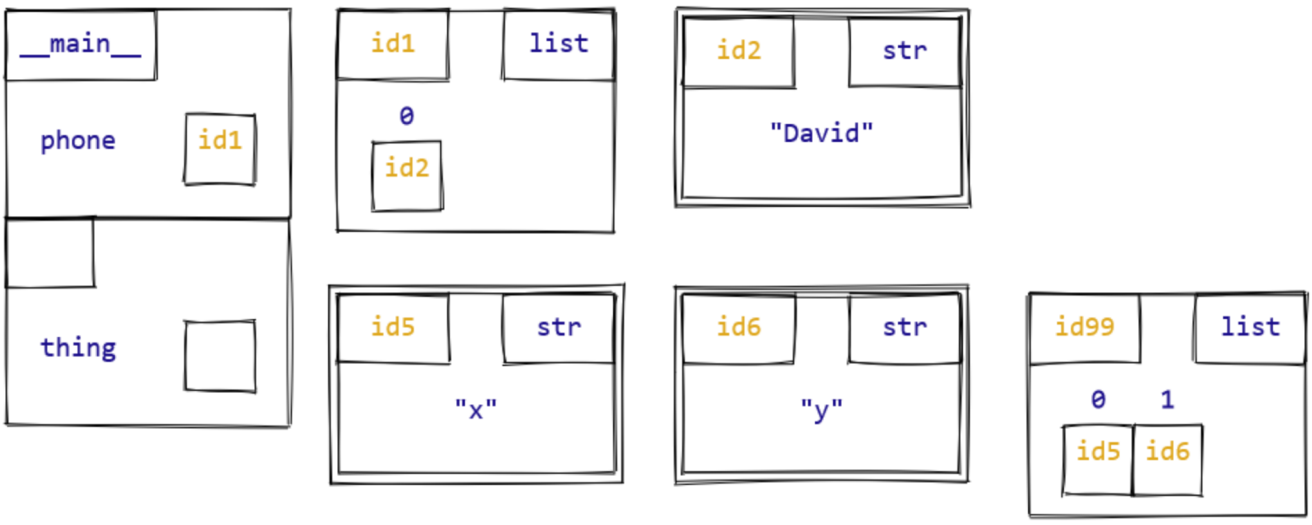
1. .

```
    def f1(thing: List[str]) -> None:
        thing = ['x', 'y'] + thing


    # In Python console
    >>> phone = ['David']
    >>> f1(phone)
    >>> phone
```



1. .

```
    def f2(thing: List[str]) -> None:
        thing.extend(['x', 'y'])


    # In Python console
    >>> phone = ['David']
    >>> f2(phone)
    >>> phone
```



CSC110 Lecture 15: Proofs

Hisbaan Noorani

October 19, 2020

Contents

1 Ex 1: Practice with proofs

1. Prove the following statement, using the definition of divisibility.

$$\forall n, d, a \in \mathbb{Z},\ d \mid n \Rightarrow d \mid an$$

We can rewrite this as: $\forall n, d, a \in \mathbb{Z}, (\exists k_1 \in \mathbb{Z}, n = dk_1) \implies (\exists k_2 \in \mathbb{Z}, an = dk_2)$

Let $n, d, a \in \mathbb{Z}$
Take $k_1 = \frac{n}{d}$
Take $k_2 = ak_1$

Assume $d \mid n$

Prove $d \mid an$:

$an = dk_2$
$an = dak_1$
$n = dk_1$
$n = dk_1 \iff an = dk_2$

Therefore we have proven $\forall n, d, a \in \mathbb{Z},\ d \mid n \Rightarrow d \mid an$, as needed.

2. Consider this statement:

$$\forall n, d, a \in \mathbb{Z},\ d \mid an \Rightarrow d \mid a \vee d \mid n$$

This statement is *False*, so here you'll disprove it.

(a) First, write the negation of this statement. You might need to review the negation rules in the Course Notes Section 3.2
$\exists n, d, a \in \mathbb{Z},\ d \mid an \wedge d \nmid a \wedge d \nmid n$

(b) Prove the negation of the statement. (By proving the statement' negation is True, you'll prove that the original statement i False.)

Let $n = 3$
Let $d = 12$
Let $a = 4$

Here we can see that $d \mid an$ is true by inputting the values of $d, a,$ and n:
$12|(3 \cdot 4)$
$12 \mid 12$
which we know is true since $\forall n \in \mathbb{R}, n \mid n$.

We now need to show that $d \nmid a$
We can show this by simply doing the division:
$= \frac{a}{d}$
$= \frac{4}{12}$
$= \frac{1}{3}$
$\notin \mathbb{Z}$. Thus $d \nmid a$ as needed

We now need to show that $d \nmid n$
$= \frac{n}{d}$
$= \frac{3}{12}$
$= \frac{1}{4}$
$\notin \mathbb{Z}$. Thus $d \nmid n$ as needed

And therefore, we have proven $\exists n, d, a \in \mathbb{Z},\ d \mid an \wedge d \nmid a \wedge d \nmid n$, as needed which implies that the original statement, $\forall n, d, a \in \mathbb{Z},\ d \mid an \Rightarrow d \mid a \vee d \mid n$, is true. ■

2 Ex 2: Primality testing

In lecture, we saw an algorithm for checking whether a number p prime that checks all of the possible factors of p between 2 and $\lfloor \sqrt{p} \rfloor$, inclusive.

We can prove that this algorithm is correct by proving the follow statement:

$$\forall p \in \mathbb{Z},\ Prime(p) \Leftrightarrow \big(p > 1(\forall d \in,\ 2 \leq d \leq \sqrt{p} \Rightarrow d \nmid \big)$$

This is a larger statement than the ones we've looked at so far, so this exercise we've broken down the proof of this statement for you complete.

Proof.
Let $p \in \mathbb{Z}$. We need to prove an if and only if, which we do dividing the proof into two parts.

2.1 Part 1: Proving that $Prime(p) \Rightarrow \big(p > 1(\forall d \in,\ 2 \leq d \leq \sqrt{p} \Rightarrow d \nmid \big)$.

1. Write down what we can **assume** in this part of the proof.

   $Prime(p) : p > 1 \wedge (\forall d \in \mathbb{N},\ d \mid p \implies d = 1 \vee d = p)$, where $p \in \mathbb{Z}$

   We assume this entire predicate.

2. To prove an AND, we need to prove that both parts are true. First, prove that $p > 1$.

   $p > 1$, as we have assumed such (as seen above)

3. Now, prove that $\forall d \in,\ 2 \leq d \leq \sqrt{p} \Rightarrow d \nmid p$.

   Assume $d \in (2, \sqrt{p})$

   This implies that $2 \leq d$ which implies $d > 1$ therfore $d \neq 1$.

   Since $p > 1$, $\sqrt{p} < p$. This implies that $d < \sqrt{p} < p$ therefore $d \neq p$

   We know that since p is prime d must be either 1 or p to divide p. Since $d \neq 1$ and $d \neq p$ we have proven $d \nmid p$ ■

2.2 Part 2: Proving that $p > 1 \wedge (\forall d \in,\ 2 \leq d \leq \sqrt{p} \Rightarrow d \nmid p) \Rightarrow \mathit{Prime}(p)$.

1. Write down what we can **assume** in this part of the proof.

   - $p > 1$
   - $\forall d \in \mathbb{N}, 2 \leq d \leq p \implies d \nmid p$

2. We need to prove that $\mathit{Prime}(p)$, which expands into $p > 1 \wedge (\forall d \in,\ d \mid p \Rightarrow d = 1 \vee d = p)$.

   First, prove that $p > 1$.

   We have proven $p > 1$ by assumption.

3. Now for the proof of $\forall d \in \mathbb{N},\ d \mid p \Rightarrow d = 1 \vee d = p$. Start by writing the appropriate proof header, introducing the variable d and assumption about d.

   Assume

   $\forall d_1 \in \mathbb{N}, d_1 \mid p$

   OR

   $\forall d_1 \in \mathbb{N}, \exists k \in \mathbb{Z}, p = kd_1$

4. Use the **contrapositive** of a part of your original assumption. What can you conclude about d?

   $\forall d_1 \in \mathbb{N} d_1 \mid p \implies d_1 < 2 \vee d_1 > \sqrt{p}$

   From this contrapositive, we can conlude that $d_1 < 2$ or $d_1 < \sqrt{p}$

5. Using the cases from the previous part, prove that $d = 1 \vee d = p$.

   Case 1: $d_1 < 2$.

   > $d_1 \in \mathbb{N} \wedge d_1 < 2$, then $d_1 = 0$ or $d_1 = 1$
   > But $0 \nmid p$, because $p > 1$, $d \neq 0$
   > Therefore $d_1 = 1$

   Case 2: $d_1 < \sqrt{p}$

   > $p = d_1 k$
   > $k < \sqrt{p}$, but k also divides p which means that $k < 2$ therefore $k = 1$
   > $d_1 = p$

   Therefore in both cases, either $d_1 = p$ or $d_1 = 1$ ■

3 Additional Exercises

1. Prove the following statement, which extends the first statement in Exercise 1.

$$\forall n, m, d, a, b \in \mathbb{Z},\ d \mid n \wedge d \mid m \Rightarrow d \mid (an + bm)$$

2. *Disprove* the following statement, which is very similar to the one you proved in Exercise 2.

$$\forall p \in \mathbb{Z},\ \mathit{Prime}(p) \Leftrightarrow \big(p > 1 \wedge (\forall d \in,\ 2 \leq d < \sqrt{p} \Rightarrow d \nmid p)\big).$$

CSC110 Lecture 16: Greatest Common Divisor

Hisbaan Noorani

October 20, 2020

Contents

1 Ex 1: Property of the greatest common divisor

For your reference, here is the definition of greatest common divisor.

> Let $x, y, d \in \mathbb{Z}$. We say that d is a **common divisor** of x and y when d divides x and d divides y.
> We say that d is the **greatest common divisor** of x and y when it the largest number that is a common divisor of x and y, or 0 when x and y are both 0.
>
> We can define the function $\gcd : \mathbb{Z} \times \mathbb{Z} \to \mathbb{N}$ as the function which takes numbers x and y, and returns their greatest common divisor.

And here is the Quotient-Remainder Theorem, slightly modified to handle negative integers.

> *Quotient-Remainder Theorem.* Let $n \in \mathbb{Z}$ and $d \in \mathbb{Z}$. If $d \neq 0$ then there exist $q \in \mathbb{Z}$ and $r \in \mathbb{N}$ such that $n = qd + r$ and $0 \leq r < d$. Moreover, these q and r are *unique* for a given n and d.
>
> We say that q is the **quotient** when n is divided by d, and that r is the **remainder** when n is divided by d. We use $n \ \% \ d$ to denote the remainder of n divided by d.

And finally, here is the Divisibility of Linear Combinations Theorem that we covered in lecture today.

> *Divisibility of Linear Combinations.* Let $m, n, d \in \mathbb{Z}$. If d divides m and d divides n, then d divides every linear combination of m and n.

1. Prove the following statement: For all $a, b, c \in \mathbb{Z}$, $b \neq 0$ and $c \mid a$ and $c \mid b$, then $c \mid a \ \% \ b$.

   $\forall a, b, c \in \mathbb{Z}, b \neq 0 \land c \mid a \land c \mid b \implies c \mid a \ \% \ b$

   Tip: introduce variables q and r using the Quotient-Remainder theorem in your proof.

   Let $a, b, c \in \mathbb{Z}$

   Assume $b \neq 0,\ c \mid a,\ c \mid b$

   By QRT, $\exists q \in \mathbb{Z}, a = qb + r \implies r = a - qb$

   r is a linear combination of a and b

   Since $c \mid a$ and $c \mid b$, then by DLC, $c \mid r$ which means $c | a \ \% \ b$ ■

2. Prove the following statement: For all $a, b \in \mathbb{Z}$, if $b \neq 0$ then $\gcd(b, a \ \% \ b) \mid a$.

   $\forall a, b \in \mathbb{Z}, b \neq 0 \implies \gcd(b, a \ \% \ b) \mid a$

   Tip: introduce variables q and r using the Quotient-Remainder theorem in your proof.

   Let $a, b \in \mathbb{Z}$

   Assume $b \neq 0$

   Let $r = a \ \% \ b$, $d = \gcd(b, r)$

   By QRT, $\exists q \in \mathbb{Z}, a = qb + r$

   Since $d = \gcd(b, r)$, then by definition, $d \mid b \wedge d \mid r$

   Since $d \mid b \wedge d \mid r$, then by DLC $d \mid qb + r$

   Therefore $d \mid a$ as needed. ■

3. Suppose we have three numbers $a, b, c \in \mathbb{Z}$, and we know that $b \neq 0$. If we want to prove that $c = \gcd(a, b)$, what would we need to prove, using the above definition of gcd?

   (a) $c \mid a$

   (b) $c \mid b$

   (c) $\forall e \in \mathbb{N}, \ (e \mid a \wedge e \mid b) \implies e \leq c$

2 Ex 2: The Extended Euclidean Algorithm

We left off our discussion of the *Extended Euclidean Algorithm* in lecture with the following code:

```python
def gcd_extended(a: int, b: int) -> Tuple[int, int, int]:
    """Return the gcd of a and b, and integers p and q such that

    gcd(a, b) == p * a + b * q.

    >>> gcd_extended(10, 3)
    (1, 1, -3)
    """
    x, y = a, b

    # Initialize px, qx, py, and qy
    px, qx = 1, 0
    py, qy = 0, 1

    while y != 0:
        # Loop invariants
        assert math.gcd(a, b) == math.gcd(x, y)  # L.I. 1
        assert x == px * a + qx * b              # L.I. 2
        assert y == py * a + qy * b              # L.I. 3

        q, r = divmod(x, y)  # quotient and remainder when a is divided by b

        # Update x and y
        x, y = y, r

        # Update px, qx, py, and qy
        px, qx, py, qy = py, qy, px - q * py, qx -q * qy
```

```
28
29      return (x, px, qx)
```

1. Recall that the loop invariants must hold for the initial values of the loop variables. Given this, how should we initialize `px`, `qx`, `py`, and `qy`? (*Hint*: use the simplest possible numbers.)

   Write your answer below, or directly in the `gcd_extended` code.

```
1  px, qx = 1, 0
2  py, qy = 0, 1
```

2. Inside the loop body, we need to figure out how to update the new variables `px`, `qx`, `py`, and `qy`. Now in order to derive the updates for `px`, `qx`, `py`, and `qy`, we need to do a bit of math.

   For an arbitrary iteration of the while loop:

   • *Let*x_0, y_0, px_0, qx_0, px_0, py_0 be the values of the variables `x`, `y`, `px`, `qx`, `py`, `qx`, and `qy` at the *start* of the iteration.

   • x_1, y_1, px_1, qx_1, py_1, qy_1 be the values of these variables at the *end* of the iteration.

   • q and r be the values of the variables `q` and `r` during the iteration.

   (a) What do the loop invariants 2 and 3 tell you about the relationships between x_0, y_0, px_0, qx_0, py_0, qy_0? (This is what we can assume to be True at the start of the loop iteration.)
   $x_0 = (px_0)(a) + (qx_0)(b)$
   $y_0 = (py_0)(a) + (qy_0)(b)$

   (b) What do the loop invariants tell you about the desired relationships between x_1, y_1, px_1, qx_1, py_1, qy_1? (This is what we want to be True at the end of the loop iteration.)
   $x_1 = (px_1)(a) + (qx_1)(b)$
   $y_1 = (py_1)(a) + (qy_1)(b)$

   (c) From the Quotient-Remainder Theorem, what is the relationship between x_0, y_0, q, and r?
   $x_0 = q \cdot y_0 + r$

   (d) What are the values of x_1 and y_1 in terms of x_0, y_0, q, and/or r?
   $x_1 = y_0$
   $y_1 = r$

   (e) What should px_1 and qx_1 equal to satisfy the invariant for x_1, using only the "0" variables?
   $px_1 = py_1$
   $qx_1 = qy_1$

   (f) (This is the hardest part) Given your answers from earlier parts, what should py_1 and qy_1 equal in order to satisfy the invariant for y_1, using only the "0" variables?
   Hint: You'll need to do a calculation to start with your invariant in (b) and replace all of the "1" variables with "0" variables.
   $py_1 = px_0 - q \cdot py_0$
   $qy_1 = qx_0 - p \cdot qy_0$

3. Using your answers to 2(e) and 2(f), complete the code for the Euclidean algorithm to update the variables `px`, `qx`, `py`, and `qy` inside the loop body. You should then be able to run the code with all of the loop invariants passing.

```
1  px, qx, py, qy = py, qy, px - q * py, qx -q * qy
```

Congratulations, you've just completed a derivation of the Extended Euclidean Algorithm!

CSC110 Lecture 17: Modular Arithmetic

Hisbaan Noorani

October 21, 2020

Contents

For your reference, here is the definition of modular equivalence.

> Let $a, b, n \in \mathbb{Z}$, with $n \neq 0$. We say that a **is eqvuialent to** b **modulo** n when $n \mid a - b$. In this case, we write $a \equiv b \pmod{n}$.

1 Ex 1: Modular arithmetic practice

1. Expand the statement $14 \equiv 9 \pmod{5}$ into a statement using the divisibility predicate. Is this statement True or False?

   $= 5 \mid (14 - 9)$

   $= 5 \mid 5$

   This is true ✓

2. Expand the statement $9 \equiv 4 \pmod{3}$ into a statement using the divisibility predicate. Is this statement True or False?

   $3 \mid (9 - 4$

   $3 \mid 5$

   This is false. $3 \neq 5 \cdot k$ for any integer k.

3. Prove the following statement using *only* the definitions of divisibility and modular equivalence (and no other statements/theorems):

   WTS $\forall a, b, c \in \mathbb{Z},\ \forall n \in \mathbb{Z}^+, a \equiv b \pmod{n} \Rightarrow ca \equiv cb \pmod{n}$

   Let $a, b, c \in \mathbb{Z}$

   Let $n \in \mathbb{Z}^+$

   Assume $a \equiv b \pmod{n}$. This implies $n \mid a - b$. This again implies $a - b = np_1, p \in \mathbb{Z}$

   Prove $ca \equiv cb \pmod{n}$.

   We can rewrite this as: $n \mid ca - cb$

   This means:

   $ca - cb = np_2, p_2 \in \mathbb{Z}$

   $c(a - b) = ncp_1, p1 \in \mathbb{Z}$

$a - b = np_1 \in \mathbb{Z}$. We have arrived at our assumption.

We have thus proven that $ca \equiv cb \pmod{n}$ as needed. ■

2 Ex 2: Modular division

Recall that last class, we implemented the following function:

```
def extended_gcd(a: int, b: int) -> Tuple[int, int, int]:
    """Return the gcd of a and b, and integers p and q such that
    gcd(a, b) == p * a + b * q.

    >>> extended_gcd(10, 3)
    (1, 1, -3)
    """
    ...
```

This class, we proved that for any $n \in \mathbb{Z}^+$ and $a \in \mathbb{Z}$, a has an inverse modulo n as long as $\gcd(a, n) = 1$. The proof we wrote can be turned into an algorithm for actually computing this modular inverse. To solidify your knowledge of this proof, complete the following function *using* `extended_gcd` *as a helper*. Make sure to include the appropriate precondition(s) based on the statement of the theorem!

```
def modular_inverse(a: int, n: int) -> int:
    """Return the inverse of a modulo n, in the range 0 to n - 1 inclusive.

    Preconditions:
        - extended_gcd(a, n)[0] == 1

    >>> modular_inverse(10, 3)  # 10 * 1 is equivalent to 1 modulo 3
    1
    >>> modular_inverse(3, 10)  # 3 * 7 is equivalent to 1 modulo 10
    7
    """
    gcd, p, q = extended_gcd(a, n)

    assert gcd == 1

    if p > 0:
        return p
    else:
        return n + p

    # You could also use range(0, n - 1) here to get p here
    # by testing every one until one works. I would have
    # done it that way but mario's solution looked good so...
```

3 Ex 3: Exponentiation and order

Consider modulo 5, which has the possible remainders $0, 1, 2, 3, 4$. In each table, fill in the value for remainder b, where $0 \leq b < 5$, that makes the modular equivalence statement in each row True. The first table is done for you. Use Python as a calculator if you would like to. (Or write a comprehension to calculate them all at once!)

1. Powers of 2.

Power of 2	Value for b
$2^1 \equiv b \pmod 5$	2
$2^2 \equiv b \pmod 5$	4
$2^3 \equiv b \pmod 5$	3
$2^4 \equiv b \pmod 5$	1
$2^5 \equiv b \pmod 5$	2
$2^6 \equiv b \pmod 5$	4

2. Powers of 3.

Power of 3	Value for b
$3^1 \equiv b \pmod 5$	3
$3^2 \equiv b \pmod 5$	4
$3^3 \equiv b \pmod 5$	2
$3^4 \equiv b \pmod 5$	1
$3^5 \equiv b \pmod 5$	3
$3^6 \equiv b \pmod 5$	4

3. Powers of 4.

Power of 4	Correct value for b
$4^1 \equiv b \pmod 5$	4
$4^2 \equiv b \pmod 5$	1
$4^3 \equiv b \pmod 5$	4
$4^4 \equiv b \pmod 5$	1
$4^5 \equiv b \pmod 5$	4
$4^6 \equiv b \pmod 5$	1

4. Using the tables above, write down the *order* of 2, 3, and 4 modulo 5:

n	$\text{ord}_5(n)$
2	4
3	4
4	2

4 Additional Exercises

1. Using only the definition of divisibility and the definition of congruence modulo n, prove the following statements.

   (a) $\forall a, b, c, d \in \mathbb{Z},\ \forall n \in \mathbb{Z}^+,\ a \equiv b \pmod n \wedge c \equiv d \pmod n \Rightarrow a + c \equiv b + d \pmod n$

   (b) $\forall a, b \in \mathbb{Z},\ \forall n \in \mathbb{Z}^+,\ (0 \leq a < n) \wedge (0 \leq b < n) \wedge (a \equiv b \pmod n) \Rightarrow a = b.$

2. Implement the following function, which is the modular analog of division. Use your `modular_inverse` function from above. Once again, figure out what the necessary precondition(s) are for this function.

```
def modular_divide(a: int, b: int, n: int) -> int:
    """Return an integer k such that ak = b (mod n).

    The return value k should be between 0 and n-1, inclusive.

    Preconditions:

    >>> modular_divide(7, 6, 11)  # 7 * 4 is equivalent to 6 modulo 11
    4
    """
```

CSC110 Lecture 18: Introduction to Cryptography

Hisbaan Noorani

October 26, 2020

Contents

1 Exercise 1: The One-Time Pad Cryptosystem

1. Suppose we want to encrypt the plaintext `'david'` using the one-time pad cryptosystem and the secret key `'mario'`. Fill in the table below to come up with the encrypted ciphertext. You may find the following useful:

```
>>> [ord(char) for char in 'david']
[100, 97, 118, 105, 100]
>>> [ord(char) for char in 'mario']
[109, 97, 114, 105, 111]
```

message char	ord of message char	key char	ord of key char	ord of ciphertext char	ciphertext char
`'d'`	100	`'m'`	109	81	`'Q'`
`'a'`	97	`'a'`	97	66	`'B'`
`'v'`	118	`'r'`	114	104	`'h'`
`'i'`	105	`'i'`	105	82	`'R'`
`'d'`	100	`'o'`	111	83	

2. Next, implement the one-time pad cryptosystem by completing the following two functions `encrypt_otp` and `decrypt_otp`. Some tips/hints:

   - The implementation is quite similar to the Caesar cipher from Section 7.1. - Remember that you can use `ord` and `chr` to convert back and forth between characters and numbers.
   - `%` has higher precedence than `+/-`, so you'll probably need to do `(a + b) % n` instead of `a + b % n`.

```
def encrypt_otp(k: str, plaintext: str) -> str:
    """Return the encrypted message of plaintext using the key k with the
    one-time pad cryptosystem.

    Precondtions:
        - len(k) >= len(plaintext)
        - all({ord(c) < 128 for c in plaintext})
        - all({ord(c) < 128 for c in k})

    >>> encrypt_otp('david', 'HELLO')
```

```
    ',&B53'
    """
    ciphertext = ''

    for i in range(len(plaintext)):
        ciphertext = ciphertext + chr((ord(plaintext[i]) + ord(k[i])) % 128)

    return ciphertext


def decrypt_otp(k: str, ciphertext: str) -> str:
    """Return the decrypted message of ciphertext using the key k with the
    one-time pad cryptosystem.

    Precondtions:
        - all({ord(c) < 128 for c in ciphertext})
        - all({ord(c) < 128 for c in k})

    >>> decrypt_otp('david', ',&B53')
    'HELLO'
    """
    plaintext = ''

    for i in range(len(ciphertext)):
        plaintext = plaintext + chr((ord(ciphertext[i]) - ord(k[i])) % 128)

    return plaintext
```

3. Check if you can get the original plaintext message back when using your `encrypt_otp` and `decrypt_otp` functions:

```
>>> plaintext = 'David'
>>> key = 'Mario'
>>> ciphertext = encrypt_otp(key, plaintext)
>>> decrypt_otp(key, ciphertext) == plaintext
True
```

2 Exercise 2: The Diffie-Hellman key exchange algorithm

We discussed in lecture how the Diffie-Hellman key exchange is computationally secure. But it's important to remember that computational security is not the same as theoretical security. Let's implement a *brute-force* algorithm for an eavesdropper to take the p, g, $g^a \% p$, and $g^b \% p$ values that Alice and Bob communicate from the algorithm, and uses this to determine the shared secret key.

Your algorithm should try to recover one of the exponents a or b simply by try all possible values: $\{1, 2, \ldots, p-1\}$. This is computationally inefficient in practice when p is chosen to be extremely large. But how quickly can we do it with small prime numbers (e.g., 23 and 2)?

```
def break_diffie_hellman(p: int, g: int, g_a: int, g_b: int) -> int:
    """Return the shared Diffie-Hellman secret key obtained from the eavesdropped information.

    Remember that the secret key is (g ** (a * b)) % p, where a and b are the secret exponents
    chosen by Alice and Bob. You'll need to find at least one of a and b to compute the secret
```

```
    key.

    Preconditions:
        - p, g, g_a, and g_b are the values exhanged between Alice and Bob
          in the Diffie-Hellman algorithm

    >>> p = 23
    >>> g = 2
    >>> g_a = 9  # g ** 5 % p
    >>> g_b = 8  # g ** 14 % p
    >>> break_diffie_hellman(p, g, g_a, g_b)  # g ** (5 * 14) % p
    16
    """
    possible_powers_a = []
    possible_powers_b = []

    # NOTE: You could also use a while loop and exit after the first a
    #       and then use another one for the first b
    for i in range(1, p):
        if g_a = (g ** i) % p:
            possible_powers_a.append(i)
        if g_b = (g ** i) % p:
            possible_powers_b.append(i)

    a = possible_powers_a[0]
    b = possible_powers_b[0]

    return (g ** (a * b)) % p
```

CSC110 Lecture 19: Public-Key Cryptography and the RSA Cryptosystem

Hisbaan Noorani

October 27, 2020

Contents

For your reference, here is one key definition and the two main theorems about modular exponentiation that we'll use today.

(*Fermat's Little Theorem*) Let $p, a \in \mathbb{Z}$ and assume p is prime and that $p \nmid a$. Then $a^{p-1} \equiv 1 \pmod{p}$.
We define the function $\varphi : \mathbb{Z}^+ \to \mathbb{N}$, called the **Euler totient function** (or **Euler phi function**), as follows:

$$\varphi(n) = \big|\{a \mid a \in \{1, \ldots, n-1\}, \text{ and } \gcd(a, n) = 1\}\big|.$$

We have the following formulas for special cases of $\varphi(n)$:

- For all primes $p \in \mathbb{Z}$, $\varphi(p) = p - 1$.
- For all *distinct* primes $p, q \in \mathbb{Z}$, $\varphi(pq) = (p-1)(q-1)$.

(*Euler's Theorem*). For all $a \in \mathbb{Z}$ and $n \in \mathbb{Z}^+$, if $\gcd(a, n) = 1$ then $a^{\varphi(n)} \equiv 1 \pmod{n}$.

1 Exercise 1: Reviewing modular exponentiation

1. Let $a, p \in \mathbb{Z}$ and assume that p is prime that $\gcd(a, p) = 1$. Using Fermat's Little Theorem, simplify each of the following expressions modulo p by reducing it to 1 or an expression of the form a^e, where the exponent e is positive and as small as possible. We've done the first one for you.

   Note: start out with $a^{p-1} \equiv 1 \pmod{p}$ and try to manipulate it to become what you want.

Power of a	Simplified expression modulo p
a^{p-1}	1
a^p	a
a^{2p-2}	1
a^{2p}	a^{2p}
a^{p^2-1}	a^{p-1}
a^{p^2}	a^p

2. let $p = 23$ and $q = 5$.

   (a) What is $\varphi(pq)$?

$$\begin{aligned}\varphi(pq) &= (23-1)(5-1) \\ &= 88\end{aligned}$$

(b) Using euler's theorem, calculate each of the following remainders (modulo $pq = 115$). we have done the first row for you (note that $(p-1)(q-1) = 88$ – keep this number in mind).
Note: start out with $2^{88} \equiv 1 \pmod{115}$ and try to manipulate it to become what you want.

power of 2	remainder modulo $pq = 115$
2^{88}	1
2^{89}	2
2^{176}	1
2^{180}	16
2^{880}	1
2^{8801}	2

2 Exercise 2: Reviewing the RSA Cryptosystem

1. The following parts get you to manually trace through the steps of the RSA cryptosystem. The calculations themselves are pretty straightforward, we just want you to review the algorithm and practice all of the steps!

   (a) Suppose we start with the primes $p = 23$ and $q = 5$. What are n and $\varphi(n)$?
   $n = 115$

   $$\begin{aligned}\varphi(n) &= \varphi(pq) \\ &= (23-1)(5-1) \\ &= 88\end{aligned}$$

   (b) Suppose $e = 3$. Find the corresponding value for d such that $ed \equiv 1 \pmod{\varphi(n)}$. (You can just use trial and error here, or the `modular_inverse` function from last week!)
   $d = 59$

   (c) What are the RSA private and public keys for these choices of p, q, and e?
   RSA private $= (p, q, d) = (25, 5, 59)$
   RSA public $= (n, e) = (115, 3)$

   (d) Suppose you want to encrypt the number 77 using the public key. What is the resulting "ciphertext" (the encrypted number)? You can use Python as a calculator to answer this.

   $$\begin{aligned}c &= m^e \ \% \ n \\ &= 77^3 \ \% \ 115 \\ &= 98\end{aligned}$$

   (e) Verify that if you decrypt this ciphertext with the private key, you get back the original number 77.

   $$\begin{aligned}m' &= c^d \ \% \ n \\ &= 98^{59} \ \% \ 115 \\ &= 77\end{aligned}$$

2. The following are some conceptual questions about the RSA algorithm to check your understanding of this algorithm.

   (a) Why does the key generation phase require that $\gcd(e, \varphi(n)) = 1$?
   This gives us the number of numbers that are coprime to n. In order to find the modular inverse, d, the $\gcd(e, \varphi(n)$ must be 1.

(b) We know that picking $e = 1$ satisfies $\gcd(e, \varphi(n)) = 1$. Yet why is $e = 1$ not a good choice of e?
Any number d would satisfy $ed \equiv 1 \pmod{\varphi(n)}$.
Making $e = 1$ would make it very easy to compute d:
$ed \equiv 1 \pmod{\varphi(n)}$
$d = 1$

(c) When we discussed encrypting a message, we said that the message had to be in the range $\{1, 2, \ldots, n-1\}$ (where the n is from the public key) Why do we not allow numbers larger than $n - 1$ to be encrypted?
This would introduce ambiguity. Ther would be more than one possible solution when you're decrypting that would me mathematically correct but not actually correct.

CSC110 Lecture 20: More on Cryptography

Hisbaan Noorani

October 28, 2020

Contents

1 Exercise 1: Implementing the RSA cryptosystem

Now that we've covered the theory behind the RSA cryptosystem, let's see how to implement it in Python! You'll do this in three steps: implementing a function that generates an RSA key pair, then a function to encrypt a number, and finally a function to decrypt a number. (Note that just like the RSA algorithm itself, the first function is the hardest. If you get stuck, skip to the encryption/decryption functions, which are simpler!)

```python
import random
import math
from typing import Tuple


def rsa_generate_key(p: int, q: int) -> \
    Tuple[Tuple[int, int, int], Tuple[int, int]]:
    """Return an RSA key pair generated using primes p and q.

    The return value is a tuple containing two tuples:
    1. The first tuple is the private key, containing (p, q, d).
    2. The second tuple is the public key, containing (n, e).

    Preconditions:
        - p and q are prime
        - p != q

    Hints:
        - If you choose a random number e between 2 and $\varphi(n)$, there isn't a guarantee
          that $gcd(e, \varphi(n)) = 1$. You can use the following pattern to keep picking
          random numbers until you get one that is coprime to $\varphi(n)$.

              e = ... # Pick an initial choice
              while math.gcd(e, ___) > 1:
                  e = ... # Pick another random choice

        - You can re-use the functions we developed last week to compute the modular inverse.
    """
    n = p * q
    phi_n = (p - 1) * (q - 1)

```

```
    e = 2

    while math.gcd(e, phi_n) != 1:
        e += 1

    d = modular_inverse(e, phi_n)

    return ((p, q, d), (n, e))


def rsa_encrypt(public_key: Tuple[int, int], plaintext: int) -> int:
    """Encrypt the given plaintext using the recipient's public key.

    Preconditions:
    - public_key is a valid RSA public key (n, e)
    - 0 < plaintext < public_key[0]
    """
    n = public_key[0]
    e = public_key[1]

    return plaintext ** e % n


def rsa_decrypt(private_key: Tuple[int, int, int]  ciphertext: int) -> int:
    """Decrypt the given ciphertext using the recipient's private key.

    Preconditions:
    - private_key is a valid RSA private key (p, q, d)
    - 0 < ciphertext < private_key[0] * private_key[1]
    """
    d = private_key[2]

    return ciphertext ** d % (p * q)
```

CSC110 Lecture 21: Asymptotic Notation for Function Growth

Hisbaan Noorani

November 02, 2020

Contents

1 Exercise 1: Practice with Big-O

Note: In Big-O expressions, it will be convenient to just write down the "body" of the functions rather than defining named functions all the time. We'll always use the variable n to represent the function input, and so when we write "$n \in \mathcal{O}(n^2)$," we really mean "the functions defined as $f(n) = n$ and $g(n) = n^2$ satisfy $f \in \mathcal{O}(g)$."

As a reminder, here is the formal definition of what "g is Big-O of f" means:

$$g \in \mathcal{O}(f) : \ \exists c, n_0 \in \mathbb{R}^+, \ \forall n \in \mathbb{N}, \ n \geq n_0 \implies g(n) \leq c \cdot f(n)$$

1. Our first step in comparing different types of functions is looking at different powers of n. Consider the following statement, which generalizes the idea that $n \in \mathcal{O}(n^2)$:

$$\forall a, b \in \mathbb{R}^+, \ a \leq b \implies n^a \in \mathcal{O}(n^b)$$

   (a) Rewrite the above statement, but with the definition of Big-O expanded.
   $\forall a, b \in \mathbb{R}^+, a \leq b \implies (\exists c, n_0 \in \mathbb{R}^+ \text{ s.t. } \forall n \in \mathbb{N}, n \geq n_0 \implies n^a \leq c \cdot n^b)$

   (b) Prove the above statement. **Hint**: you can actually pick c and n_0 to both be 1, and have the proof work. Even though this is pretty simple, take the time to understand and write down the proof, as a good warm-up.
   (Pf:)
   Let $a, b \in \mathbb{R}^+$
   Assume $a \leq b$
   Take $c = 1$
   Take $n_0 = 1$
   Let $n \in \mathbb{N}$
   Assume $n \geq n_0$
   Prove $n^a \leq c \cdot n^b$. Since $c = 1$, we can simplify this to $n^a \leq n^b$
   We want to prove this for every $n \geq n_0$
   We know that $\forall x \in \mathbb{R}^+, a \leq b \implies x^a \leq x^b$.
   If we take this x as n, then we know that $n^a \leq n^b$ ■

2. In this question, we'll investigate what it means to show that a function *isn't* Big-O of another.

(a) Express the statement $g \notin \mathcal{O}(f)$ in predicate logic, using the expanded definition of Big-O. (As usual, simplify so that all negations are pushed as far "inside" as possible.)

$\forall c, n_0 \in \mathbb{R}^+, \exists n \in \mathbb{N}$ s.t. $n \geq n_0 \wedge g(n) > c \cdot f(n)$

(b) Prove that for all positive real numbers a and b, if $a > b$ then $n^a \notin \mathcal{O}(n^b)$.

Hints:

- For all positive real numbers x and y, $x > y \log x > \log y$.
- In your negated statement from part (a), you should find that you can only pick the value of n. Work backwards from the target inequalities to find what value of n works.

$\forall a, b \in \mathbb{R}^+,\ a > b \implies (\forall c, n_0 \in \mathbb{R}^+, \exists n \in \mathbb{N}$ s.t. $n \geq n_0 \wedge n^a > c \cdot n^b)$

Pf:

Let $a, b \in \mathbb{R}^+$

Assume $a > b$

Let $c, n_0 \in \mathbb{R}^+$

Take $n \geq \lceil n_0 + c^{1/(a-b)} \rceil$

Prove $n \geq n_0$ and $n^a > c \cdot n^b$

Since we took n_0 + a positive real number, we know that $n \geq n_0$, and thus we have proved the first part of the statement.

I won't do it here, but if you plug in the value for n that we have chosen, it will work out with the second condition. ■

2 Exercise 2: Omega and Theta

1. Consider the following statement: for all $a, b \in \mathbb{R}^+$, $an^2 + b \in \Theta(n^2)$.

(a) Expand this statement using the definition of Theta involving c_1 and c_2.

$\forall a, b \in R^+, \exists c_1, c_2, n_0 \in \mathbb{R}^+$ s.t. $\forall n \in \mathbb{N}^+, n \geq n_0 \implies c_1 n^2 \leq an^2 + b \leq c_2 n^2$

(b) You should see two different inequalities that you need to prove: $c_1 n^2 \leq an^2 + b$ and $an^2 + b \leq c_2 n^2$. Which of these variables are universally-quantified? Which are existentially quantified?

Universally Quantified ($\forall$): a, b, n

Existentially Quantified ($\exists$): c_1, c_2, n_0

(c) Choose values for the existentially-quantified variables from (b) to make the two inequalities true (you might need to do some rough calculations). Remember that when choosing values for these variables, you can define them in terms of any variables that appear to the *left* of them in the statement you wrote in part (a).

Take $c_1 = a$

Take $c_2 = a + b$

Take $n_0 = 1$

(d) Prove your statement from part (a).

Pf:

Let $a, b \in \mathbb{R}^+$

Take $c_1 = a$

Take $c_2 = a + b$

Take $n_0 = 1$

Let $n \in \mathbb{N}$

Assume $n \geq n_0$

Prove $c_1 n^2 \leq an^2 + b$ and $an^2 + b \leq c_2 n^2$

i. Prove $c_1 n^2 \leq an^2 + b$

$$\begin{aligned} c_1 n^2 &\leq an^2 + b \\ (a)n^2 &\leq an^2 + b \qquad (\text{since } c_1 = a) \\ 0 &\leq b \end{aligned}$$

Since b is a positive real number, this is true. $\blacksquare_1$

ii. Prove $an^2 + b \leq c_2 n^2$

$$\begin{aligned} an^2 + b &\leq c_2 n^2 \\ an^2 + b &\leq (a+b)n^2 \qquad (\text{since } c_2 = a + b) \\ an^2 + b &\leq an^2 + bn^2 \\ b &\leq bn^2 \end{aligned}$$

Since $n \in \mathbb{N}$, this is true. $\blacksquare_2$

Since we have shown that both parts of the conclusion of the implication are true, we have proven $an^2 + b \in \Theta(n^2)$, as needed. $\blacksquare$

3 Additional Exercises

1. One slight oddness about the definition of Big-O is that it treats all logarithmic functions "the same." Your task in this question is to investigate this, by proving the following statement:

$$\forall a, b \in \mathbb{R}^+,\ a > 1 \wedge b > 1 \implies \log_a n \in \Theta(\log_b n)$$

We won't ask you to expand the definition of Theta, but if you aren't quite sure, then we highly recommend doing so before attempting even your rough work!

Hint: look up the "change of base rule" for logarithms, if you don't quite remember it.

2. Prove that for all $a, b \in \mathbb{R}^+$, if $a < b$ then $a^n \in \mathcal{O}(b^n)$ and $a^n \notin \Omega(b^n)$.

3. As we discuss in the Course Notes, *constant functions* like $f(n) = 100$ will play an important role in our analysis of running time of algorithms. For now let's get comfortable with the notation.

   (a) Let $g : \mathbb{N} \to \mathbb{R}^{\geq 0}$. Show how to express the statement $g \in \Theta(1)$ by expanding the definition of Theta. (Here 1 refers t othe constant function $f(n) = 1$.)

   (b) Prove that $100 + \frac{77}{n+1} \in \Theta(1)$.

CSC110 Lecture 22: Properties of Asympyotic Growth and Basic Algorithm Running Time

Hisbaan Noorani

November 03, 2020

Contents

1 Exercise 1: Properties of asymptotic growth

1. Recall the following definition:

> Let $f, g : \mathbb{N} \to \mathbb{R}^{\geq 0}$. We can define the **sum of f and g** as the function $f + g : \mathbb{N} \to \mathbb{R}^{\geq 0}$ such that
>
> $$(f + g)(n) = f(n) + g(n), \qquad \text{for } n \in \mathbb{N}$$
>
> For example, if $f(n) = 2n$ and $g(n) = n^2 + 3$, then $(f + g)(n) = 2n + n^2 + 3$.

Consider the following statement:

$$\forall f, g : \mathbb{N} \to \mathbb{R}^{\geq 0},\ g \in \mathcal{O}(f) \implies f + g \in \mathcal{O}(f)$$

In other words, if g is Big-O of f, then $f + g$ is no bigger than just f itself, asymptotically speaking.

(a) Rewrite this statement by expanding the definition of Big-O (twice!). Use subscripts to help keep track of the variables. This is a good exercise in writing a complex statement in predicate logic, and will help with writing the proof in the next part.
$\forall f, g : \mathbb{N} \to \mathbb{R}^{\geq 0}, [\exists c_1, n_1 \in \mathbb{R} \text{ s.t. } \forall n \in \mathbb{N}, n \geq n_1 \implies g(n) \leq c_1 \cdot f(n)]$
$\implies [\exists c_2, n_2 \in \mathbb{R} \text{ s.t. } \forall n \in \mathbb{N}, n \geq n_2 \implies (f + g)(n) \leq c_2 \cdot f(n)]$

(b) Prove this statement.
Hint: This is an implication, so you're going to *assume* that $g \in \mathcal{O}(f)$, and you want to *prove* that $f + g \in \mathcal{O}(f)$.
(Pf:)
Let $f, g : \mathbb{N} \to \mathbb{R}^{\geq 0}$
Assume $g \in \mathcal{O}(f)$. This means $\exists c_1, n_1 \in \mathbb{R}$ s.t. $\forall n \in \mathbb{N}, n \geq n_1 \implies g(n) \leq c_1 \cdot f(n)$
Prove $f + g \in \mathcal{O}(f)$. This means $\exists c_2, n_2 \in \mathbb{R}$ s.t. $\forall n \in \mathbb{N}, n \geq n_2 \implies (f + g)(n) \leq c_2 \cdot f(n)$
Take $n_2 = n_1$
Take $c_2 = c_1 + 1$
Let $n \in \mathbb{N}$
Assume $n \geq n_2$

Prove $(f+g)(n) \leq c_2 \cdot f(n)$
Since $n_2 = n_1$, we know $n \geq n_1$ thus $g(n) \leq c_1 \cdot f(n)$.

$$
\begin{aligned}
(f+g)(n) &\leq c_2 \cdot f(n) \\
f(n) + g(n) &\leq (c_1 + 1) \cdot f(n) && \text{(since } c_2 = c_1 + 1\text{)} \\
f(n) + g(n) &\leq c_1 \cdot f(n) + f(n) \\
g(n) &\leq c_1 \cdot f(n) \quad \blacksquare && \text{(subtract } f(n) \text{ from both sides)}
\end{aligned}
$$

We have arrived at our assumption, $g(n) \leq c_1 \cdot f(n)$, and thus we have proven $f + g \in \mathcal{O}(f)$ as needed.

2 Exercise 2: Analysing running time (for loops)

Analyse the running time of each of the following functions, in terms of their input length n. Keep in mind these three principles for doing each analysis:

- For each for loop, determine the *number of iterations* and the *number of steps per iteration*.
- When you see statements in sequence (one after the other), determine the number of steps for each statement separately, and then add them all up.
- When dealing with nested loops, start by analyzing the inner loop first (the total steps of the inner loop will influence the steps per iteration of the outer loop).

1. Number of iterations: n

   Number of steps per iteration: 2

   $RT_{f_1} = 2 \cdot n$
   $RT_{f_1} \in \Theta(n)$

```
1    def f1(numbers: List[int]) -> None:
2        for number in numbers:
3            print(number * 2)
```

1. $\underline{1^{\text{st}} \text{ loop}}$

   Number of iterations: n

   Number of steps per iteration: 2

   $\underline{2^{\text{nd}} \text{ loop}}$

   Number of iterations: 10

   Number of steps per iteration: 3

   $RT_{f_2} = 2 \cdot n + 3 \cdot 10$
   $RT_{f_2} \in \Theta(n)$

```
1    def f2(numbers: List[int]) -> int:
2        sum_so_far = 0
3        for number in numbers:
4            sum_so_far = sum_so_far + number
5
6        for i in range(0, 10):
7            sum_so_far = sum_so_far + number * 2
8
9        return sum_so_far
```

2. Inner loop

   Number of iterations: n

   Number of steps per iteration: 2

   Outer loop

   Number of Iterations: $n^2 + 5$

   Number of steps per iteration: $2n$

   $RT_{f_3} = 2n^3 + 10n$

   $RT_{f_3} \in \Theta(n^3)$

```
    def f3(numbers: List[int]) -> int:
        for i in range(0, len(numbers) ** 2 + 5):
            for number in numbers:
                print(number * i)
```

3 Additional exercises

Review the properties of Big-O/Omega/Theta we covered in lecture today, and try proving them! You should be able to prove all of them except (5) and (6) of the Theorem on the growth hierarchy of elementary functions.

CSC110 Lecture 23: More Running-Time Analysis

Hisbaan Noorani

November 04, 2020

Contents

1 Exercise 1: Analysing program runtime of while loops

Your task here is to **analyse the running time of each of the following functions**. Recall the technique from lecture for calculating the number of iterations a while loop takes:

1. Find i_0, the value of the loop variable at the start of the first iteration. (You may need to change the notation depending on the loop variable name.)
2. Find a pattern for i_0, i_1, i_2, etc. based on the loop body, and a formula for a general i_k, the value of the loop variable after k loop iterations, assuming that many iterations occurs.
3. Find the *smallest* value of k that makes the loop condition False. This gives you the number of loop iterations.

   You'll need to use the floor/ceiling functions to get the correct exact number of iterations.

Note: each loop body runs in constant time in this question. While this won't always be the case, such examples allow you to focus on just counting loop iterations here.

```
1 1.    def f1(n: int) -> None:
2           """Precondition: n >= 3."""
3           i = 3
4           while i < n:
5               print(i)
6               i = i + 5
```

$i_0 = 3$

$i_k = 3 + 5k$

$n \leq 3 + 5k$

$k \geq \frac{n-3}{5}$

$RT_{f_1} = \lceil \frac{n-3}{5} \rceil$

$RT_{f_1} \in \Theta(n)$

12.
```python
def f2(n: int) -> None:
    """Preconditions: n > 0 and n % 10 == 0."""
    i = 0
    while i < n:
        print(i)
        i = i + (n // 10)
```

$i_0 = 0$

$i_k = k \cdot \lfloor \frac{n}{10} \rfloor$

$n \leq k \cdot \lfloor \frac{n}{10} \rfloor$

$k \geq 10$

$RT_{f_2} = 10$

$RT_{f_2} \in \Theta(1)$

13.
```python
def f3(n: int) -> None:
    """Precondition: n >= 5."""
    i = 20
    while i < n * n:
        print(i)
        i = i + 3
```

$i_0 = 20$

$i_k = 20 + 3k$

$i_k \geq n^2$

$20 + 3k \geq n^2$

$k \geq \frac{n^2 - 20}{3}$

$RT_{f_3} = 3 + \left\lceil \frac{n^2 - 20}{3} \right\rceil$

$RT_{f_3} \in \Theta(n^2)$

14.
```python
def f4(n: int) -> None:
    """Precondition: n >= 2."""
    i = 2
    while i < n:
        print(i)
        i = i * i
```

$i_0 = 2$

$i_k = 2^{2^k}$

$i_k \geq n$

$$
\begin{aligned}
2^{2^k} &\geq n \\
\log_2(2^{2^k}) &\geq \log_2(n) \\
2k \cdot \log_2(2) &\geq \log_2(n) \\
2^k &\geq \log_2(n) \\
\log_2(2^k) &\geq \log_2(\log_2(n)) \\
k \cdot \log_2(2) &\geq \log_2(\log_2(n)) \\
k &\geq \log_2(\log_2(n))
\end{aligned}
$$

$RT_{f_4} = \lceil \log_2(\log_2(n)) \rceil$

$RT_{f_4} \in \Theta(\log(\log(n))$

2 Exercise 2: Analysing nested loops

Each of the following functions takes as input a non-negative integer and performs at least one loop. Remember, to analyse the running time of a nested loop:

1. First, determine an expression for the exact cost of the inner loop for a fixed iteration of the outer loop. This may or may be the same for each iteration of the outer loop.
2. Then determine the total cost of the outer loop by adding up the costs of the inner loop from Step 1. Note that if the cost of the inner loop is the same for each iteration, you can simply multiply this cost by the total number of iterations of the outer loop. Otherwise, you'll need to set up and simplify an expression involving summation notation (Σ).
3. Repeat Steps (1) and (2) if there is more than one level of nesting, starting from the innermost loop and working your way outwards. Your final result should depend *only* on the function input, not any loop counters.

You'll also find the following formula helpful:

$$\sum_{i=0}^{n} i = \frac{n(n+1)}{2}$$

```
11.     def f5(n: int) -> None:
2           """Precondition: n >= 0"""
3           i = 0
4           while i < n: # iterates \ciel{n/5} times
5               for j in range(0, n): # iterates n times
6                   print(j)
7               i = i + 5
```

Inner loop: iterates n times

Outer loop: iterates $\lceil \frac{n}{5} \rceil$ times (Same analysis as f_1)

$RT_{f_5} = n \cdot \lceil \frac{n}{5} \rceil$

$RT_{f_5} \in \Theta(n^2)$

12.
```python
def f6(n: int) -> None:
    """Precondition: n >= 4"""
    i = 4
    while i < n:
        j = 1
        while j < n:
            j = j * 3
        for k in range(0, i):
            print(k)
        i = i + 1
```

Inner loop 1: iterates $\lceil \log_3(n) + 1 \rceil$ times.

$j_0 = 1$

$j_k = 3^{j-1}$

$j_k \geq n$

$3^{j-1} \geq n$

$\log_3(3^{j-1}) \geq \log_3(n)$

$(j-1) \cdot \log_3(3) \geq \log_3(n)$

$j \geq \log_3(n) + 1$

Inner loop 2: iterates i times

Outer loop: iterates $n - 4$ times

$i_0 = 4$

$i_k = 4 + k$

$i_k \geq n$

$4 + k \geq n$

$k \geq n - 4$

I'm not really sure how to get this one... sorry. Maybe I'll get it later.

13.
```python
def f7(n: int) -> None:
    """Precondition: n >= 0"""
    i = 0
    while i < n: # iterates \ciel{n/4} times
        j = n
        while j > 0: # iterates
            for k in range(0, j): # iterates j times
                print(k)
            j = j - 1
        i = i + 4
```

Innermost loop: iterates j times

Middle loop: iterates n times

Innermost and Middle loop: $\sum_{i=0}^{n} j = \frac{n(n+1)}{2}$

Outermost loop: iterates $\lceil \frac{n}{4} \rceil$ times

$RT_{f_7} = \lceil \frac{n}{4} \rceil \cdot \frac{n(n+1)}{2}$

$RT_{f_7} \in \Theta(n^3)$

3 Additional exercises

1. Analyse the running time of the following function.

```
def f8(n: int) -> None:
    """Precondition: n >= 1"""
    i = 1
    while i < n:
        j = 0
        while j < i:
            j = j + 1

        i = i * 2
```

Hint: the actual calculation for this function is a little trickier than the others. You may need a formula from Appendix C.1 Summations and Products. Note: you can look up a formula for "sum of powers of 2" or "geometric series" for the analysis in this question. This analysis is trickier than the others.

2. Consider the following algorithm:

```
def subsequence_sum(lst: List[int]) -> int:
    n = len(lst)
    max_so_far = 0
    for i in range(0, n):              # Loop 1: i goes from 0 to n-1
        for j in range(i, n):          # Loop 2: j goes from i to n-1
            s = 0
            for k in range(i, j + 1):  # Loop 3: k goes from i to j
                s = s + lst[k]
            if max_so_far < s:
                max_so_far = s
    return max_so_far
```

Determine the Theta bound on the running time of this function (in terms of n, the length of the input list). For practice, do not make any approximations on the *number of iterations* of any of the three loops; that is, your analysis should actually calculate the total number of iterations of the innermost k-loop across all iterations of the outer loop. Go slow! Treat this is as a valuable exercise in performing calculations with summation notation.

You may find the following formulas helpful (valid for all $n, a, b \in^+$):

$$\sum_{i=0}^{n} i = \frac{n(n+1)}{2}, \qquad \sum_{i=0}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}, \qquad \sum_{i=a}^{b} f(i) = \sum_{i'=0}^{b-a} f(i'+a)$$

CSC110 Lecture 24: Analyzing Built-In Data Type Operations

Hisbaan Noorani

November 17, 2020

Contents

1 Exercise 1: Running time of list operations

Each of the following Python functions takes a list as input. Analyse each one's running time in terms of n, the size of its input.

```
1 1.    def f1(nums: List[int]) -> None:
2           list.insert(nums, 0, 10000) # n steps
```

$RT_{f_1} \in \Theta(n)$

```
1 2.    def f2(nums: List[int]) -> None:
2           for i in range(0, 100):        # 100 steps
3               list.append(nums, 10000)   # 1 step
```

$RT_{f_2} \in \Theta(100)$

```
1 3.    def f3(nums: List[int]) -> None:
2           for i in range(0, 100):            # 100 steps
3               list.insert(nums, 0, 10000)    # n^2 steps
```

Note: the length of `nums` changes at each iteration, and so the running time of `list.insert` does as well!

i	shifts
0	$n+0$
1	$n+1$
2	$n+2$
...	...
99	$n+99$

$100n + \frac{99(99+1)}{2} \in \Theta(n)$

$RT_{f_3} \in \Theta(n)$

```
14.     def f4(nums: List[int]) -> None:
2           n = len(nums)                   # 1 step
3           for i in range(0, n * n):       # n^2 steps
4               list.insert(nums, 0, i)     # n^3 steps
```

i	shifts
0	$n+0$
1	$n+1$
2	$n+2$
...	...
n^2-1	$n+n^2-1$

$$(n^2)(n) + \frac{(n^2-1)(n^2-1+1)}{2} \in \Theta(n^4)$$

$RT_{f_4} \in \Theta(n^4)$

2 Exercise 2: Running-time analysis with multiple parameters

Each of the following functions takes more than one list as input. Analyse their running time in terms of the size of their inputs; do not make any assumptions about the relationships between their sizes.

```
11.     def f5(nums1: List[int], nums2: List[int]) -> None:
2           for num in nums2:               # n_2 steps
3               list.append(nums1, num)     # 1 step
```

(Let n_1 be the size of `nums1` and n_2 be the size of `nums2`.)

$RT_{f_5} \in \Theta(n_2)$

```
12.     def f6(nums1: List[int], nums2: List[int]) -> None:
2           for num in nums2:                   # n_2 steps
3               list.insert(nums1, 0, num)      # n_1 * n_2 steps
```

(Let n_1 be the size of `nums1` and n_2 be the size of `nums2`.)

$n_2 \cdot n_1 \cdot \frac{(n_2)(n_2+1)}{2} \in \Theta(n_1 \cdot n_2 + (n_2)^2)$

$RT_{f_6} \in \Theta(n_1 \cdot n_2 + (n_2)^2)$

```
13.     def f7(nested_nums: List[List[int]]) -> None:
2           for nums in nested_nums:            # n steps
3               list.insert(nums, 0, 10000)     # m steps
```

(Let n be the length of `nested_nums`, and assume each inner list has length m.)

$RT_{f_7} \in \Theta(n \cdot m)$

3 Exercise 3: Sets, dictionaries, and data classes

Analyse the running time of each of the following functions.

11.
```
def f8(nums: Set[int]) -> bool:
    return 1 in nums or 2 in nums
```

$RT_{f_8} \in \Theta(1)$

12.
```
def f9(num_map: Dict[int, int]) -> None:
    for num in num_map:
        num_map[num] = num_map[num] + 1
```

Let $n =$ `len(num_map)`

$RT_{f_8} \in \Theta(n)$

13.
```
def f10(grades: Dict[str, List[int]], new_grades: Dict[str, int]):
    for course in new_grades: # n^2 steps
        if course in grades:  # 1 step (for the entire bock)
            list.append(grades[course], new_grades[course])
        else:
            grades[course] = [new_grades[course]]
```

Let $n_1 =$ `len(grades)`

Let $n_2 =$ `len(new_grades)`

$RT_{f_{10}} \in \Theta(n_2)$

14.
```
def f11(people: List[Person]) -> int:
    """Precondition: people != []"""
    max_age_so_far = -math.inf            # 1 step

    for person in people:                 # n steps
        if person.age > max_age_so_far:   # 1 step (for the entire block)
            max_age_so_far = person.age

    return max_age_so_far                 # 1 step
```

(Assume the `math` module has been imported, and we've defined a `Person` data class with four attributes, including `age`.)

Let $n =$ `len(people)`

$RT_{f_{11}} \in \Theta(n)$

4 Additional exercises

Analyse the running time of each of the following functions.

11.
```
def extra1(nums: List[int]) -> None:
    for i in range(0, len(nums)):  # n iterations
        nums[i] = 0                # 1 step
```

Let n be the length of `nums`.

$RT_{e_1} \in \Theta(n)$

12.
```python
def extra2(nums: List[int]) -> None:
    for i in range(0, len(nums)):  # n iterations
        list.pop(nums)             # 1 step
```

Let n be the length of `nums`.

$RT_{e_2} \in \Theta(n)$

13.
```python
def extra3(nums: List[int]) -> None:
    for i in range(0, len(nums)):  # n iterations
        list.pop(nums, 0)          # n steps
```

Note: the length of `nums` changes at each iteration, and so the running time of `list.pop` does as well!

Let n be the length of `nums`.

14.
```python
def extra4(nums1: List[int], nums2: List[int]) -> None:
    for i in range(0, len(nums1)):             # n_1 iterations
        for j in range(0, len(nums2)):         # n_2 iterations
            nums1[i] = nums1[i] + nums2[j]     # 2 steps
```

Let n_1 be the size of `nums1` and n_2 be the size of `nums2`

$RT_{e_4} \in \Theta(n_1 \cdot n_2)$

15.
```python
def extra5(nested_nums: List[List[int]]) -> None:
    for i in range(0, len(nested_nums)):               # n steps
        if i == 0:
            for j in range(0, len(nested_nums[0])):    # m steps
                nested_nums[i][j] = 0
        else:
            nested_nums[i][0] = 0
```

Let n be the length of `nested_nums`, and assume every inner list has length m.

$RT_{e_5} \in \Theta(n + m)$

16.
```python
def extra6(nums: Set[int]) -> List[int]:
    new_nums = []                              # 1 step

    for num in nums:                           # n iterations
        list.insert(new_nums, 0, num ** 2)     # n steps

    return new_nums                            # 1 step
```

Let n be the length of `nums`.

$RT_{e_6} \in \Theta(n^2)$

```python
def extra7(nums: List[int]) -> Dict[int, int]:
    counts_so_far = {}                                  # 1 step

    for num in nums:                                    # n iterations
        if num in counts_so_far:                        # 1 step (in for dict)
            counts_so_far[num] = counts_so_far[num] + 1 # 2 steps
        else:
            counts_so_far[num] = 1                      # 1 step

    return counts_so_far                                # 1 step
```

Let n be the length of `nums`.

$RT_{e8} \in \Theta(n)$

CSC110 Lecture 25: Worst-Case Running Time Analysis

Hisbaan Noorani

November 18, 2020

Contents

1 Exercise 1: Worst-case running time analysis practice

Consider the following function, which has an early return:

```
def are_disjoint_lists(nums1: List[int], nums2: List[int]) -> bool:
    """Return whether nums1 and nums2 are disjoint sets of numbers."""
    for x in nums1:     # n steps
        if x in nums2:  # n steps
            return False

    return True
```

Note: For your analysis in this exercise, assume both input lists have the same length n.

1. Find a tight upper bound (Big-O) on the worst-case running time of `are_disjoint_lists`. By "tight" we mean it should be possible to prove the same lower bound (Omega), but we're not asking you to do it until the next question.

   Use phrases like *at most* to indicate inequalities in your analysis.

   Let $n = |\text{nums1}| = |\text{nums2}|$

   One iteration takes *at most* $n + 1$ steps when x is the last element in the list

   The for loop iterates n times.

   $WC_f = n \cdot (n + 1) = n^2 + n \in \wr(n^2)$

2. Prove a matching lower bound on the worst-case running time of `are_disjoint_lists`. Remember that this means finding an input family whose asymptotic running time is the same as the upper bound you found in Question 1.

   Let $n = |\text{nums1}| = |\text{nums2}|$

   Let nums1 and nums2 be lists that are disjoint.

   $\forall x \in \text{nums1}, x \notin \text{nums2}$

   Then, a single iteration of the for loop takes at least n steps.

   The loop iterates n times.

   Therefore $WC_f \in \Omega(n^2)$

3. Using Questions 1 and 2, conclude a tight Theta bound on the worst-case running time of `are_disjoint_lists`.
   Since we've shown that $WC_f \in \wr(n^2)$ and $WC_f \in \Omega(n^2)$, we can say that $WC_f \in \Theta(n^2)$

2 Exercise 2: Lists vs. sets

Now consider the following function, which is the same as the previous one, but operates on sets instead of lists:

```
def are_disjoint_sets(nums1: Set[int], nums2: Set[int]) -> bool:
    """Return whether nums1 and nums2 are disjoint sets of numbers."""
    for x in nums1:     # n steps
        if x in nums2:  # 1 step
            return False

    return True
```

Note: all parts of this question explores a few variations of the analysis you did in Exercise 1. To save time, don't rewrite your full analysis. Just describe the parts that would change, and the final bound that you get.

1. Analyse the worst-case running time of `are_disjoint_sets`, still assuming that the two input sets have the same length.

   Big-O

   One iteration takes at most 1 step.

   The loop iterates n times.

   $WC_f \in \wr(n)$

   Omega

   Let $n = |\text{nums1}| = |\text{nums2}|$

   Let nums1 and nums2 be sets that are disjoint.

   $\forall x \in \text{nums1}, x \notin \text{nums2}$

   Then a single iteration of the loop takes at least 1 step.

   The loop iterates n times.

   Therefore $WC_f \in \Omega(n)$

   Theta

   Since we've shown that $WC_f \in \wr(n)$, and $WC_f \in \wr(n)$, we can say that $WC_f \in \Theta(n)$

2. Now let's consider what happens if we take the assumption that `nums1` and `nums2` have different lengths. For this question, let n_1 be the length of `nums1` and n_2 be the length of `nums2`.

   (a) What would the worst-case running time of `are_disjoint_lists` be in this case, in terms of n_1 and/or n_2?
   Using similar analysis to *Ex1*, we can say that $WC_f \in \Theta(n_1 \cdot n_2)$

   (b) What would the worst-case running time of `are_disjoint_sets` be, in terms of n_1 and/or n_2?
   Using similar analysis to *Ex2 Q1*, we can say that $WC_f \in \Theta(n_1)$

   (c) What would the worst-case running time of `are_disjoint_sets` be, in terms of n_1 and/or n_2, if we switched the `nums1` and `nums2` in the function body?
   Switching nums1 and nums2 would result in a $WC_f \in \Theta(n_2)$

   (d) Can you write an implementation of `are_disjoint_sets` whose worst-case running time is $\Theta(\min(n_1, n_2))$?

```python
def are_disjoint_sets(nums1, nums2) -> bool:
    """Return whether the sets are disjoint or not, with efficiency Theta(min(n_1, n_2))"""
    if(len(nums1) > len(nums2)):
        for x in nums2:
            if x in nums1:
                return False
        return True
    else:
        for x in nums1:
            if x in nums2:
                return False
        return True
```

3 Additional exercises: Substring search

Here is an algorithm which takes two strings and determines whether the first string is a substring of the second.

```python
def substring(s1: str, s2: str) -> bool:
    """Return whether s1 is a substring of s2."""
    for i in range(0, len(s2) - len(s1)):                                    # n2 - n1 iterations
        # Check whether s1 == s2[i: i + len(s1) - 1]
        found_match = all(s1[j] == s2[i + j] for j in range(0, len(s1)))    # n1 iterations

        # If a match has been found, stop and return True.
        if found_match:
            return True

    return False
```

1. Let n_1 represent the length of `s1` and n_2 represent the length of `s2`. Find a tight asymptotic upper bound on the worst-case running time of `substring` in terms of n_1 and/or n_2.

   The worst case for this function would be if `s2` was not a substring of `s1`. This means that the function would not return until it reaches the end of the string.

   This would lead to a worst case running time $\in \Theta(n_1 \cdot (n_2 - n_1))$

2. Find an input family whose running time matches the upper bound you found in Question 1. You may assume that $n_1 \mid n_2$ (this may make it a bit easier to define an input family).

   Hint: you can pick `s1` to be a string of length n_1 that just repeats the same character n_1 times.

   If you take the string `s1` to be one character repeated n_1 times, and you take the string `s2` to be a different character repeatead n_2 times, then the two strings will be separate and distinct. This means that `s2` would not be a substring of `s2`. This means that the function will go through its entire loop and will then after doing all of that, return `False`.

CSC110 Lecture 26: Abstract Data Types and Stacks

Hisbaan Noorani

November 23, 2020

Contents

1 Exercise 1: Using Stacks

Before we get into implementing stacks, we are going to put ourselves in the role of a stack *user*, and attempt to implement the following top-level function (*not* method):

```
def size(s: Stack) -> int:
    """Return the number of items in s.

    >>> s = Stack()
    >>> size(s)
    0
    >>> s.push('hi')
    >>> s.push('more')
    >>> s.push('stuff')
    >>> size(s)
    3
    """


if __name__ = '__main__':
    s = Stack()

    for i in range(0, 100):
        s.push(i)

    # should print out 100.
    print(str(size(s)) + 'is the size of your stack!')
```

1. Each of the following four implementations of this function has a problem. For each one, explain what the problem is.

   Note: some of these functions may seem to work correctly, but do not exactly follow the given docstring because they mutate the stack `s` as well!

(a)
```
def size(s: Stack) -> int:
    """Return the number of items in s.
    """
    count = 0
    for _ in s:
        count = count + 1
    return count
```

A stack does not have 'elements' in the same way a list does so you cannot itterate throught them one like this.

(b)
```
def size(s: Stack) -> int:
    """Return the number of items in s.
    """
    count = 0
    while not s.is_empty():
        s.pop()
        count = count + 1
    return count
```

This does not return the stack to its original state.

(c)
```
def size(s: Stack) -> int:
    """Return the number of items in s.
    """
    return len(s._items)
```

The `_items` variable is a private instance attribute. You should not use this as an implementation can change, changing the exitence or behaviour of this private instance attribute.

(d)
```
def size(s: Stack) -> int:
    """Return the number of items in s.
    """
    s_copy = s
    count = 0
    while not s_copy.is_empty():
        s_copy.pop()
        count += 1
    return count
```

When you write `s_copy = s`, you are coppying the memory address. This means that when you modify `s_copy`, you're modifying `s` as well.

2. Write a correct implementation of the `size` function. You can use the same approach as (b) from the previous question, but use a second, temporary stack to store the items popped off the stack.

```
def size(s: Stack) -> int:
    """Return the number of items in s.

    >>> s = Stack()
    >>> size(s)
    0
    >>> s.push('hi')
```

```
    >>> s.push('more')
    >>> s.push('stuff')
    >>> size(s)
    3
    """
    temp_stack = Stack()

    counter = 0

    while not s.is_empty():
        temp_stack.push(s.pop())
        counter += 1

    while not temp_stack.is_empty():
        s.push(temp_stack.pop())

    return counter
```

2 Exercise 2: Stack implementation and running-time analysis

1. Consider the implementation of the Stack we just saw in lecture:

```
class Stack1:
    """A last-in-first-out (LIFO) stack of items.

    Stores data in first-in, last-out order. When removing an item from the
    stack, the most recently-added item is the one that is removed.

    >>> s = Stack1()
    >>> s.is_empty()
    True
    >>> s.push('hello')
    >>> s.is_empty()
    False
    >>> s.push('goodbye')
    >>> s.pop()
    'goodbye'
    """
    # Private Instance Attributes:
    #   - _items: The items stored in the stack. The end of the list represents
    #     the top of the stack.
    _items: list

    def __init__(self) -> None:
        """Initialize a new empty stack.
        """
        self._items = []

    def is_empty(self) -> bool:
        """Return whether this stack contains no items.
        """
        # can also say ~~ not self._items ~~ instead of this.
```

```
        return self._items == []

    def push(self, item: Any) -> None:
        """Add a new element to the top of this stack.
        """
        self._items.append(item)

    def pop(self) -> Any:
        """Remove and return the element at the top of this stack.

        Preconditions:
            - not self.is_empty()
        """
        return self._items.pop()
```

Analyse the running times of the `Stack1.push` and `Stack1.pop` operations in terms of n, the size of the stack.

$RT_{\text{push}} \in \Theta(1)$

$RT_{\text{pop}} \in \Theta(1)$

2. Our implementation of `Stack1` uses the back of its list attribute to store the top of the stack. In the space below, complete the implementation of `Stack2`, which is very similar to `Stack1`, but now uses the *front* of its list attribute to store the top of the stack.

```
class Stack2:
    """A last-in-first-out (LIFO) stack of items.

    Stores data in first-in, last-out order. When removing an item from the
    stack, the most recently-added item is the one that is removed.

    >>> s = Stack2()
    >>> s.is_empty()
    True
    >>> s.push('hello')
    >>> s.is_empty()
    False
    >>> s.push('goodbye')
    >>> s.pop()
    'goodbye'
    """
    # Private Instance Attributes:
    #   - _items: The items stored in the stack. The end of the list represents
    #     the top of the stack.
    _items: list

    def __init__(self) -> None:
        """Initialize a new empty stack.
        """
        self._items = []


    def is_empty(self) -> bool:
        """Return whether this stack contains no items.
```

```
        """
        # can also say ~~ not self._items ~~ instead of this.
        return self._items == []


    def push(self, item: Any) -> None:
        """Add a new element to the top of this stack.
        """
        self._items.insert(0, item)


    def pop(self) -> Any:
        """Remove and return the element at the top of this stack.

        Preconditions:
            - not self.is_empty()
        """
        return self._items.pop(0)
```

3. Analyse the running time of the `Stack2.push` and `Stack2.pop` methods.

   $RT_{\text{push}} \in \Theta(n)$

   $RT_{\text{pop}} \in \Theta(n)$

4. Based on your answers to Questions 1 and 3, which stack implementation should we use, `Stack1` or `Stack2`?

   We should use `Stack1` as it has a lower running time for the pop method, even though the push method remains the same.

3 Additional exercises

Each of the following functions takes at least one stack argument. Analyse the running time of each function *twice*: once assuming it uses `Stack1` as the stack implementation, and again using `Stack2`. (We use the type annotation `Stack` as a placeholder for either `Stack1` or `Stack2`.)

11.
```
def extra1(s: Stack) -> None:
    s.push(1)
    s.pop()
```

Stack1

$RT_{S_1} \in \Theta(n)$

Stack2

$RT_{S_2} \in \Theta()$

12.
```
def extra2() -> None:
    s = Stack1()  # Or, s = Stack2()

    for i in range(0, 5):
        s.push(i)
```

Stack1

$RT_{S_1} \in \Theta(1)$

Stack2

$RT_{S_2} \in \Theta(n)$

13.
```
    def extra3(s: Stack, k: int) -> None:
        """Precondition: k >= 0"""
        for i in range(0, k):
            s.push(i)
```

Stack1

$RT_{S_1} \in \Theta(1)$

Stack2

$RT_{S_2} \in \Theta(n)$

4. s1 starts as a stack of size n, and s2 starts as an empty stack

```
    def extra4(s1: Stack) -> None:
        s2 = Stack1()  # Or, s2 = Stack2()

        while not s1.is_empty():
            s2.push(s1.pop())

        while not s2.is_empty():
            s1.push(s2.pop())
```

Stack1

$RT_{S_1} \in \Theta(n)$

Stack2

$RT_{S_2} \in \Theta(n^2)$

CSC110 Lecture 27: Queues and Priority Queues

Hisbaan Noorani

November 24, 2020

Contents

1 Exercise 1: Queue implementation and running time analysis

Consider the implementation of the Queue ADT from lecture:

```
class Queue:
    """A first-in-first-out (FIFO) queue of items.

    Stores data in a first-in, first-out order. When removing an item from the
    queue, the most recently-added item is the one that is removed.

    >>> q = Queue()
    >>> q.is_empty()
    True
    >>> q.enqueue('hello')
    >>> q.is_empty()
    False
    >>> q.enqueue('goodbye')
    >>> q.dequeue()
    'hello'
    >>> q.dequeue()
    'goodbye'
    >>> q.is_empty()
    True
    """
    # Private Instance Attributes:
    #   - _items: The items stored in this queue. The front of the list represents
    #             the front of the queue.
    _items: list

    def __init__(self) -> None:
        """Initialize a new empty queue."""
        self._items = []

    def is_empty(self) -> bool:
        """Return whether this queue contains no items.
```

```
        """
        return self._items == []

    def enqueue(self, item: Any) -> None:
        """Add <item> to the back of this queue.
        """
        self._items.append(item)

    def dequeue(self) -> Any:
        """Remove and return the item at the front of this queue.

        Preconditions:
            - not self.is_empty()
        """
        return self._items.pop(0)
```

1. Complete the following table of running times for the operations of our Queue implementation. Your running times should be Theta expressions in terms of n, the number of items stored in the queue. Briefly justify each running time in the space below the table; no formal analysis necessary.

   (*Note*: equality comparison to an empty list is a constant-time operation.)

Method	Θ Runtime
`__init__`	$RT \in \Theta(1)$
`is_empty`	$RT \in \Theta(1)$
`enqueue`	$RT \in \Theta(1)$
`dequeue`	$RT \in \Theta(n)$

2. You should notice that at least one of these operations takes $\Theta(n)$ time—not great! Could we fix this by changing our implementation to use the *back* of the Python list to store the front of the queue?

```
class QueueReversed:
    """A first-in-first-out (FIFO) queue of items.

    Stores data in a first-in, first-out order. When removing an item from the
    queue, the most recently-added item is the one that is removed.

    >>> q = Queue()
    >>> q.is_empty()
    True
    >>> q.enqueue('hello')
    >>> q.is_empty()
    False
    >>> q.enqueue('goodbye')
    >>> q.dequeue()
    'hello'
    >>> q.dequeue()
    'goodbye'
    >>> q.is_empty()
    True
    """
    # Private Instance Attributes:
    #   - _items: The items stored in this queue. The front of the list represents
```

```
    #             the front of the queue.
    _items: list

    def __init__(self) -> None:
        """Initialize a new empty queue."""
        self._items = []

    def is_empty(self) -> bool:
        """Return whether this queue contains no items.
        """
        return self._items == []

    def enqueue(self, item: Any) -> None:
        """Add <item> to the back of this queue.
        """
        self._items.insert(0, item)

    def dequeue(self) -> Any:
        """Remove and return the item at the front of this queue.

        Preconditions:
            - not self.is_empty()
        """
        return self._items.pop()
```

No, if we switched it to treat the back of the list as the front of the queue, then while the `dequeue` function would be $\Theta(1)$, the `enqueue` function would be $\Theta(n)$. Either way, one of the functions has to be $Theta(n)$.

2 Exercise 2: Priority Queues

1. Complete the following implementation of the Priority Queue ADT, which uses a private attribute that is an *unsorted list of tuples* (pairs of (priority, value)) to store the elements in the collection.

```
class PriorityQueueUnsorted:
    """A queue of items that can be dequeued in priority order.

    When removing an item from the queue, the highest-priority item is the one
    that is removed.

    >>> pq = PriorityQueueUnsorted()
    >>> pq.is_empty()
    True
    >>> pq.enqueue(1, 'hello')
    >>> pq.is_empty()
    False
    >>> pq.enqueue(5, 'goodbye')
    >>> pq.enqueue(2, 'hi')
    >>> pq.dequeue()
    'goodbye'
    """
    # Private Instance Attributes:
    #   - _items: A list of the items in this priority queue.
```

```
    #             Each element is a 2-element tuple where the first element is
    #             the priority and the second is the item.

    _items: List[Tuple[int, Any]]

    def __init__(self) -> None:
        """Initialize a new and empty priority queue."""
        self._items = []


    def is_empty(self) -> bool:
        """Return whether this priority queue contains no items.
        """
        return not self._items

    def enqueue(self, priority: int, item: Any) -> None:
        """Add the given item with the given priority to this priority queue.
        """
        self._items.append((priority, item))

    def dequeue(self) -> Any:
        """Return the element of this priority queue with the highest priority.

        Preconditions:
            - not self.is_empty()
        """
        greatest_priority = 0
        pop_index = 0
        for i in range len(self._items):
            if self._items[i][0] > greatest_priority:
                greatest_priority = self._items[i][0]
                pop_index = i

        return self._item.pop(pop_index)[1]
```

2. Complete the following table of running times for the operations of your `PriorityQueueUnsorted` implementation. Your running times should be Theta expressions in terms of n, the number of items stored in the priority queue. Briefly justify each running time in the space below the table; no formal analysis necessary.

Method	Θ Runtime
`__init__`	$RT \in \Theta(1)$
`is_empty`	$RT \in \Theta(1)$
`enqueue`	$RT \in \Theta(1)$
`dequeue`	$RT \in \Theta(n)$

3 Additional exercises

1. Implement a new version of the Priority Queue ADT called `PriorityQueueSorted`, which also stores a list of `(priority, item)` pairs, except it keeps the list sorted by priority.

   Your implementation should have a running time of $\Theta(1)$ for `dequeue` and a *worst-case* running time of $\Theta(n)$ for `enqueue` (but possibly with a faster running time depending on the item and priority being inserted).

CSC110 Lecture 28: Inheritance

Hisbaan Noorani

November 25, 2020

Contents

1 Exercise 1: Inheritance

1. Please use the following questions to review the terminology we have covered so far this lecture.

   (a) What is an **abstract method**?

   A method that is declaired but does not have an implementation. The body of this method will raise a `NotImplementedError`.

   (b) What is an **abstract class**?

   A class that contains one or more abstract methods.

   (c) Consider the following Python class. Is it abstract or concrete?

```
class MyClass:

    def do_something(x: int) -> int:
        return x + 5

    def do_something_else(y: int) -> int:
        raise NotImplementedError
```

   Abstract. `MyClass.do_something` is an abstract method.

2. Consider the Stack inheritance hierarchy introduced in lecture, where the abstract class `Stack` is the parent class of both `Stack1` and `Stack2`. For each of the following code snippets in the Python console, write the output or describe the error that would occur, and explain.

   (a)

```
>>> s = Stack2()
>>> isinstance(s, Stack1)
False
```

   `Stack2` and `Stack1` are both children of `Stack` but that does not mean that if an object is an instance of `Stack2` then it is also an instance of `Stack1`. They are siblings but not twins.

(b)
```
>>> s = Stack1()
>>> Stack.push(s, 'book')
>>> Stack.pop(s)
NotImplementedError
```

Since this method is not implemented in the `Stack` class, python will raise a `NotImplementedError`. We should be calling from `Stack1`.

(c)
```
>>> s = Stack()
>>> s.push('paper')
NotImplementedError
```

The `Stack` class itself does not have an implementation of its methods. It relies on its child classes so one of those should be initialized instead.

3. We have said that inheritance serves as another form of *contract*:

   - The implementor of the subclass must implement the methods from the abstract superclass.
   - Any user of the subclass may assume that they can call the superclass methods on instances of the subclass.

   What happens if we violate this contract? Once again, consider the classes `Stack` and `Stack1`. Expect this time, the method `Stack1.is_empty` is missing:

```
class Stack1(Stack):
    """..."""
    # Private Instance Attributes
    #   _items: The elements in the stack
    _items: List[Any]

    def __init__(self) -> None:
        """Initialize a new empty stack."""
        self._items = []

    def push(self, item: Any) -> None:
        """Add a new element to the top of this stack.
        """
        list.append(self._items, item)

    def pop(self) -> Any:
        """Remove and return the element at the top of this stack.

        Preconditions:
            - not self.is_empty()
        """
        return list.pop(self._items)
```

Try executing the following lines of code in the Python console—what happens?

```
>>> s = Stack1()
>>> s.push('pancake')
>>> s.is_empty()
```

This will raise a `NotImplementedError`. This class violates the contract. It does not implement the `Stack.is_empty` method so it inherits the one that is not implemented.

2 Exercise 2: Polymorphism

```
def weird(stacks: List[Stack]) -> None:
    for stack in stacks:
        if stack.is_empty():
            stack.push('pancake')
        else:
            stack.pop()
```

1. Suppose we execute the following code in the Python console:

```
>>> list_of_stacks = [Stack1(), Stack2(), Stack1(), Stack2()]
>>> list_of_stacks[0].push('chocolate')
>>> list_of_stacks[2].push('chocolate')
```

Now suppose we call `weird(list_of_stacks)`. Given the list `list_of_stacks`, write the specific `push` or `pop` method that would be called at each loop iteration. The first is done for you.

`weird` iteration	`push=/=pop` version
0	`Stack1.pop`
1	`Stack2.push`
2	`Stack1.pop`
3	`Stack2.push`

2. Write a code snippet in the Python console that results in a variable `list_of_stacks2` that, if passed to `weird`, would result in the following sequence of `push=/=pop` method calls: `Stack1.push`, `Stack2.push`, `Stack1.pop`, `Stack2.pop`.

```
>>> list_of_stacks2 = [Stack1(), Stack2(), Stack1(), Stack2()]
>>> list_of_stacks2[2].push('chocolate')
>>> list_of_stacks2[3].push('chocolate')
```

3. Create a list `list_of_stacks3` that, if passed to `weird`, would raise a `NotImplementedError` on the second loop iteration.

```
>>> list_of_stacks3 = [Stack1(), Stack]
```

3 Exercise 3: The `object` superclass and overriding methods

1. Does our `Stack` abstract class have a parent class? If so, what is it? If not, why not?

   Yes, it is the `object` class.

2. Suppose we have a variable `my_stack = Stack1()`. What information does the string representation `str(my_stack)` display?

   It will represent the type of stack, as well as the items in the stack.

   In this case, that will be `'Stack1: empty'`.

3. In the space below, override the `__str__` method for the `Stack1` class, so that the string representation matches the format shown in the docstring.

   Note: You should call `str` on each item stored in the stack.

```
class Stack1:
    _items: list

    # ... other code omitted

    def __str__(self) -> str:
        """Return a string representation of this stack.

        >>> s = Stack1()
        >>> str(s)
        'Stack1: empty'
        >>> s.push(10)
        >>> s.push(20)
        >>> s.push(30)
        >>> str(s)
        'Stack1: 30 (top), 20, 10'

        Notes:
            - because this is a method, you may access the _items attribute
            - call str on each element of the stack to get string representations
              of the items
            - review the str.join method
            - you can reverse the items in a list by calling reversed on it
              (returns a new iterable) or the list.reverse method (mutates the list)
        """
        str_so_far = 'Stack1: '
        if self.is_empty():
            str_so_far += 'empty'
        else:
            strings = [str(item) for item in self._items]
            list.reverse(strings)
            strings[0] = strings[0] + ' (top)'
            str_so_far += ', '.join(strings)

        return str_so_far
```

CSC110 Lecture 29: Object-Oriented Modelling

Hisbaan Noorani

November 30, 2020

Contents

1 Exercise 1: Designing a `Courier` data class

In this week's prep, you read about four data classes we'll use this week to model the different entities in our food delivery system: `Restaurant`, `Customer`, `Courier`, and `Order`. We gave the full design for `Restaurant` and `Order` in the Course Notes, and guided you through an implementation of `Customer` in the prep. However, we left `Courier` blank:

```
@dataclass
class Courier:
    """A person who delivers food orders from restaurants to customers.

    Instance Attributes:
        - name: name of the courier.
        - location: the current location of the lattitude.
        - order: the order currently assigned to the courier.

    Representation Invariants:
        - self.name != ''
        - -90 <= self.location[0] <= 90
        - -180 <= self.location[1] <= 180
        - self.order.courier is self or self.order.courier is None
    """
    name: str
    location: Tuple[float, float]
    order: Optional[Order] = None
```

In this exercise, you will design this class. We recommend completing this exercise in the starter file `entities.py` we've posted under Week 11 on Quercus.

1. First, we want all couriers to have these three attributes:

   - A name (which should not be empty)
   - A location (latitude and longitude, just like restaurants and customers)

- A *current order*, which is either `None` (if they have no order currently assigned to them) or an `Order` instance (if they have an order assigned to them).

  The *default value* for this attribute should be `None`—review the `Order` data class for how to set a default value for an instance attribute.

Add to the given definition of the `Courier` data class to include these three instance attributes. Make sure to include type annotations, descriptions, and representation invariants for these attributes. Also write an *example use* of this data class as a doctest example.

2. One thing to note for this design is that every `Order` instance has an associated `Courier` attribute, and every `Courier` has an associated `Order` attribute. This leads to a new representation invairant:

   - If `self` has a non-`None` current order, then that `Order` object's `courier` attribute is equal to `self`.

   Translate this representation invariant into Python code; use `is` to check for reference equality between `self` and the order's `courier`.

3. Can two `Order` objects refer to the same `Courier` instance? Why or why not?

   Yes, when a `Courier` finishes an `Order`, they can be assigned to another `Order`. Just not at the same time. You cannot have two incomplete orders refferring to the same `Courier`, only one `Courier` can be reffered to by an uncompleted (outstanding) `Order`.

4. Brainstorm two or three other instance attributes you could add to the `Courier` data class to better model "real world" food delivery systems. Pick meaningful names and type annotations for these instance attributes.

   There are no right or wrong answers here! You are practicing brainstorming a small part of object-oriented design.

   - `phone_number: str`
   - `mode_of_transport: str`

2 Exercise 2: Developing the `FoodDeliverySystem` class

In lecture we introduced the start of a new class to act as a "manager" of all the entities in the network.

```
class FoodDeliverySystem:
    """A system that maintains all entities (restaurants, customers, couriers, and orders).

    Representation Invariants:
        - self.name != ''
        - all(r == self._restaurants[r].name for r in self._restaurants)
        - all(c == self._customers[c].name for c in self._customers)
        - all(c == self._couriers[c].name for c in self._couriers)
    """
    # Private Instance Attributes:
    #   - _restaurants: a mapping from restaurant name to Restaurant object.
    #       This represents all the restaurants in the system.
    #   - _customers: a mapping from customer name to Customer object.
    #       This represents all the customers in the system.
    #   - _couriers: a mapping from courier name to Courier object.
    #       This represents all the couriers in the system.
    #   - _orders: a list of all orders (both open and completed orders).

    _restaurants: Dict[str, Restaurant]
```

```
    _customers: Dict[str, Customer]
    _couriers: Dict[str, Courier]
    _orders: List[Order]
```

Now, we're going to ask you to implement two different methods for this class. We recommend completing this exercise in the starter file `food_delivery_system.py` we've posted under Week 11 on Quercus.

1. Implement the `FoodDeliverySystem` initializer, which simply initializes all of the instance attributes to be empty collections of the appropriate type.

```
def __init__(self) -> None:
    """Initialize a new food delivery system.

    The system starts with no entities.
    """
    self._restaurants = {}
    self._customers = {}
    self._couriers = {}
    self._orders = []
```

2. Implement the `FoodDeliverySystem.add_restaurant` method, which adds a new restaurant to the system. Because the `FoodDeliverySystem` keeps track of all entities, it can check uniqueness constraints across all the restaurants—something that individual `Restaurant` instances can't check for.

```
def add_restaurant(self, restaurant: Restaurant) -> bool:
    """Add the given restaurant to this system.

    Do NOT add the restaurant if one with the same name already exists.

    Return whether the restaurant was successfully added to this system.
    """
    if restaurant.name in self._restaurants:
        return False

    self._restaurants[restaurant.name] = restaurant
    return True
```

For extra practice later, implement the analogous `add_customer` and `add_courier` methods to this class.

```
def add_customer(self, customer: Customer) -> bool:
    """Add the given customer to this system.

    Do NOT add the customer if one with the same name already exists.

    Return whether the customer was successfully added to this system.
    """
    if customer.name in self._customers:
        return False

    self._customers[customer.name] = customer
    return True
```

```python
def add_courier(self, courier: Courier) -> bool:
    """Add the given courier to this system.

    Do NOT add the courier if one with the same name already exists.

    Return whether the courier was successfully added to this system.
    """
    if courier.name in self._couriers:
        return False

    self._couriers[courier.name] = courier
    return True
```

3 Exercise 3: Handling orders

Handling new orders is more complex than the other entities, since there are two steps involved.

- First, a new order is assigned an available courier.
- Second, at a later time the order is marked as complete.

Your task for this exercise is to implement each of the two methods below. You should do this in the same file you used for Exercise 2.

Note: When choosing a particular courier to assign, you may choose how you want to make the choice: e.g., the first courier in `self._couriers` who is available, or perhaps the one that is closest to the restaurant or customer?

```python
def place_order(self, order: Order) -> bool:
    """Try to add an order to this system.

    Do NOT add the order if no couriers are available (i.e., are already assigned orders).

    - If a courier is available, add the order and assign it a courier, and return True.
    - Otherwise, do not add the order, and return False.

    Preconditions:
        - order not in self._orders
    """
    available_couriers = [courier for courier in self._couriers
                          if self._couriers[courier].order is None]

    if not available_couriers:
        return False

    # Set the courier's order
    courier = available_couriers[0]
    courier.order = order

    # Set the order's courier
    order.courier = courier
    self._orders.append(order)
    return True

```

```python
def complete_order(self, order: Order, timestamp: datetime.datetime) -> None:
    """Record that the given order has been delivered successfully at the given timestamp.

    Make the courier who was assigned this order available to take a new order.

    In addition to implementing the function, add the following preconditions:
        - the order is not already complete
        - the given timestamp is after the order's start time
          (you can use < to compare datetime.datetimes)

    Preconditions:
        - order.end_time is None
        - order.start_time < timestamp
    """
    order.end_time = timestamp
    order.courier.order = None

    # Don't do:
    # order.courier = None
    # self._orders.remove(order)
    #
    # These were not specified in the docstring and it would be useful to keep this information.
```

CSC110 Lecture 30: Discrete-Event Simulations

Hisbaan Noorani

January 14, 2021

Contents

Note: like yesterday, we've provided a starter file `events.py` for you to complete your work for today's worksheet (Exercises 1 and 2).

1 Exercise 1: Representing events

In lecture, you learned about the `Event` abstract class, used to represent a single change in the state of our food delivery system.

```
class Event:
    """An abstract class representing an event in a food delivery simulation.

    Instance Attributes:
        - timestamp: the start time of the event
    """
    timestamp: datetime.datetime

    def __init__(self, timestamp: datetime.datetime) -> None:
        """Initialize this event with the given timestamp."""
        self.timestamp = timestamp

    def handle_event(self, system: FoodDeliverySystem) -> None:
        """Mutate the given food delivery system to process this event.
        """
        raise NotImplementedError
```

1. First, please review both this class and the `NewOrderEvent` we developed together in lecture. Make sure you understand both of them (and the relationship between them) before moving on.
2. Your main task here is to implement a new event class called `CompleteOrderEvent` that represents when a courier has completed a delivery to a customer.

   Its structure should be very similar to `NewOrderEvent`, except:

   (a) Its initializer needs an explicit `timestamp` parameter (to represent when the order is completed).

   (b) The implementation of `handle_event` needs to call a different `FoodDeliverySystem` method—please review yesterday's code for this.

```python
class CompleteOrderEvent(Event):
    """An event representing when an order is delivered to a customer by a courier."""
    _order: Order

    def __init__(self, timestamp: datetime.datetime, order: Order) -> None:
        Event.__init__(self, timestamp)
        self.order = order

    def handle_event(self, system: FoodDeliverySystem) -> List[Event]:
        system.complete_order(self._order, self.timestamp)
        return []
```

2 Exercise 2: The `GenerateOrdersEvent`

Consider the `GenerateOrdersEvent` we covered in lecture (atttributes and initializer shown):

```python
class GenerateOrdersEvent(Event):
    """An event that causes a random generation of new orders.
    """
    # Private Instance Attributes:
    #   - _duration: the number of hours to generate orders for
    _duration: int

    def __init__(self, timestamp: datetime.datetime, duration: int) -> None:
        """Initialize this event with timestamp and the duration in hours.

        Preconditions:
            - duration > 0
        """
        Event.__init__(self, timestamp)
        self._duration = duration
```

Your task here is to implement its `handle_event` method, which does not mutate the given `FoodDeliverySystem`, but instead randomly generates a list of =NewOrderEvent=s using the following algorithm:

1. Initialize a variable `current_time` to be this event's `timestamp`.
2. Create a new `Order` by randomly choosing a customer and restaurant, an empty `food_items` dictionary, and the `current_time`.
3. Create a new `NewOrderEvent` based on the `Order` from Step 2, and add it to a list accumulator.
4. Increase the `current_time` by a random number of minutes, from 1 to 60 inclusive.
5. Repeat Steps 2–4 until the `current_time` is greater than the `GenerateOrderEvent`'s `timestamp` plus its `_duration` (in hours).

We've started this method for you; you only need to fill in the body of the while loop. This is good practice with the `random` module!

```python
def handle_event(self, system: FoodDeliverySystem) -> List[Event]:
    """Generate new orders for this event's timestamp and duration."""
    # Technically the lines below access a private attribute of system,
    # which is a poor practice. We'll discuss an alternate approach in class.
```

```
    customers = [system._customers[name] for name in system._customers]
    restaurants = [system._restaurants[name] for name in system._restaurants]

    events = []  # Accumulator

    current_time = self.timestamp
    end_time = self.timestamp + datetime.timedelta(hours=self._duration)

    while current_time < end_time:
        customer = random.choice(customers)
        restaurant = random.choice(restaurants)

        random_order = Order(customer, restaurant, {}, current_time)
        new_order_event = NewOrderEvent(random_order)
        events.append(new_order_event)

        current_time = current_time + datetime.timedelta(minutes=random.randint(1, 60))

    return events
```

```
def handle_event(self, system: FoodDeliverySystem) -> List[Event]:
    """Generate new orders for this event's timestamp and duration."""
    # FIXME: Create some public methods in FoodDeliverySystem to access
    #        a list of customers and a list of restaurants
    customers = [system._customers[name] for name in system._customers]
    restaurants = [system._restaurants[name] for name in system._restaurants]

    events = []  # Accumulator

    current_time = self.timestamp
    end_time = self.timestamp + datetime.timedelta(hours=self._duration)

    while current_time < end_time:
        customer = random.choice(customers)
        restaurant = random.choice(restaurants)

        random_order = Order(customer, restaurant, {}, current_time)
        new_order_event = NewOrderEvent(random_order)
        events.append(new_order_event)

        current_time = current_time + datetime.timedelta(minutes=random.randint(1, 60))
```

3 Exercise 3: Understanding the main simulation loop

Recall the main simulation loop from lecture:

```
def run_simulation(initial_events: List[Event], system: FoodDeliverySystem) -> None:
    events = EventQueueList()  # Initialize an empty priority queue of events
    for event in initial_events:
        events.enqueue(event)

```

```
6       # Repeatedly remove and process the next event
7       while not events.is_empty():
8           event = events.dequeue()
9
10          new_events = event.handle_event(system)
11          for new_event in new_events:
12              events.enqueue(new_event)
```

Your goal for this exercise is to review the three `Event` subclasses we've seen so far and see how to trace the execution of this loop.

Suppose we call `run_simulation` with a single initial event:

- type `OrderGenerateEvent`, timestamp `December 1 2020, 11:00am`, duration 1 hour

Complete the following table, showing the state of the priority queue `events` after each loop iteration. For each event, only show its class name and the time from the timestamp, not the day (all events for this example will take place on the same day). We've given an example in the first two rows.

(Note that since there's some randomness in `OrderGenerateEvent.handle_event`, we assumed that it creates *three* `NewOrderEvents` that occur at 11:00, 11:07, and 11:20.)

Loop Iteration	Events stored in `events`
0	`OrderGenerateEvent(11:00)`
1	`NewOrderEvent(11:00)`, `NewOrderEvent(11:07)`, `NewOrderEvent(11:20)`
2	`NewOrderEvent(11:07)`, `CompleteOrderEvent(11:10)`, `NewOrderEvent(11:20)`,
3	`CompleteOrderEvent(11:10)`, `CompleteOrderEvent(11:17)`, `NewOrderEvent(11:20)`
4	`CompleteOrderEvent(11:17)`, `NewOrderEvent(11:20)`
5	`NewOrderEvent(11:20)`
6	`CompleteOrderEvent(11:30)`
7	`<empty>`

While loop stops since `events` is empty

CSC110 Lecture 31: Discrete-Event Simulators

Hisbaan Noorani

December 02, 2020

Contents

1 Exercise 1: Completing `FoodDeliverySimulation`

Your task for this exercise is to review the code we covered in lecture for the `FoodDeliverySimulation` class, and then complete the two helper methods for the initializer. Again, you can complete your work here or in the starter file `simulation.py` we've posted on Quercus.

```
class FoodDeliverySimulation:
    """A simulation of the food delivery system.

    >>> simulation = FoodDeliverySimulation(datetime.datetime(2020, 11, 30), 7, 4, 100, 50)
    >>> simulation.run()
    """
    # Private Instance Attributes:
    #   - _system: The FoodDeliverySystem instance that this simulation uses.
    #   - _events: A collection of the events to process during the simulation.
    _system: FoodDeliverySystem
    _events: EventQueue

    def __init__(self, start_time: datetime.datetime, num_days: int,
                 num_couriers: int, num_customers: int,
                 num_restaurants: int) -> None:
        """Initialize a new simulation with the given simulation parameters.

        start_time: the starting time of the simulation
        num_days: the number of days that the simulation runs
        num_couriers: the number of couriers in the system
        num_customers: the number of customers in the system
        num_restaurants: the number of restaurants in the system
        """
        self._events = EventQueueList()
        self._system = FoodDeliverySystem()

        self._populate_initial_events(start_time, num_days)
        self._generate_system(num_couriers, num_customers, num_restaurants)

    def _populate_initial_events(self, start_time: datetime.datetime, num_days: int) -> None:
```

```
        """Populate this simulation's Event priority queue with GenerateOrdersEvents.

        One new GenerateOrdersEvent is generated per day for num_days,
        starting with start_time.
        Each GenerateOrdersEvent's duration is 24 hours.
        """
        # TODO: complete this method

    def _generate_system(self, num_couriers: int, num_customers: int, num_restaurants: int) -> None:
        """Populate this simulation's FoodDeliverySystem with the specified number of entities.

        You can initialize restaurants with empty menus.
        """
        for i in range(0, num_customers):
            location = _generate_location()
            customer = Customer(f'Customer {i}', location)
            self._system.add_customer(customer)

        # TODO: complete this method

    def run(self) -> None:
        """Run this simulation.
        """
        while not self._events.is_empty():
            event = self._events.dequeue()

            new_events = event.handle_event(self._system)
            for new_event in new_events:
                self._events.enqueue(new_event)
```

2 Exercise 2: Reporting statistics

As we discussed in lecture, now that we have a full `FoodDeliverySimulation` class, we can write methods to report statistics on a simulation that has already been run.

Your task is to implement the method `FoodDeliverySimulation.restaurant_order_stats`, which returns the maximum, minimum, and average number of completed orders for a single restaurant during the run of the simulation.

```
class FoodDeliverySimulation:
    ...

    def restaurant_order_stats(self) -> Dict[str, float]:
        """Return summary statistics for how many orders each restaurant received.

        The returned dictionary contains three keys:
            - 'max': the maximum number of orders made to a single restaurant
            - 'min': the minimum number of orders made to a single restaurant (can be 0)
            - 'average': the average number of orders made to a single restaurant

        Preconditions:
            - self.run() has already been called
        """
```

```
        # As we discussed yesterday, we can add a new method FoodDeliverySystem.get_restaurants()
        # instead of accessing a private attribute _restaurants.
        orders_per_restaurant = {name: 0 for name in self._system._restaurants}

        for order in self._system._orders:
            orders_per_restaurant[order.restaurant.name] += 1

        keys = list(orders_per_restaurant.keys())
        order_nums = [orders_per_restaurant[key] for key in keys]
        max_num = float(max(order_nums))
        min_num = float(min(order_nums))
        average = float(sum(order_nums) / len(order_nums))

        return {'max': max_num, 'min': min_num, 'average': average}
```