
# CS 1112: Introduction To Programming

## Exam 2 Review

Dr. Nada Basit // basit[*at*]Virginia[*dot*]edu

Spring 2024

# Friendly Reminders

- Your ***safety*** and ***comfort*** is important!
  - If you choose to wear a mask you are welcome to do so
  - *We will interpret wearing a mask as being considerate and caring of others in the classroom (not that you are sick), and realize that some may choose to mask to remain distanced*
- Be an ***active*** participant in your learning!
  You're welcome and ***encouraged*** to ask questions during class!
- If you feel ***unwell***, or think you are, please stay home
  - *We will work with you!*
  - Get some rest ☺
  - View the recorded lectures – *please allow 24-48 hours to post*
  - *Contact us!*

# Announcements

- **No new Quiz out tonight**
- ***Next homework on RegEx will be out after the exam***

- **Exam 2** is on April 8, 2024 (Monday)
  - If you have **SDAC** *time and/or distraction-free accommodations*, please **book** a time slot with SDAC to take the exam at their facility (*any time* on April 8, not another day, please!)

# Exam 2: Policies

- Closed-book, closed-notes,... closed everything ☺
- Closed PyCharm (IDE)
- No collaboration at all - *must represent your individual effort*
- Location: Taken in-class (in-person)
- Exam taken on Sherlock
- Duration: class time (1 hour and 15 minutes)
  - Please come on time!

# General

- Emphasis on content and reading / understanding code
- Only asked to write a very small amount of code, if any
- Read code and answer questions about it
- Given input, trace code, what is the output / what does the code do?
- Understand the various data structures, what they're used for, pros/cons, …
- Discuss reasons for using one data structure over the other / explain reasons why / etc.
- Given code, what might be wrong with it?
- Match or provide strings based on given code (e.g., RegEx)
- Given regular expressions recognize strings that would match (& similar questions)

# Topics

- Strings
- Lists
- Tuples
- Dictionaries
- Nested data structures
- Regular Expressions

- NOT:
  - classes

Note on cumulative material:

Computer Science topics build on one another, so material we covered earlier in the semester (e.g., for Exam 1) *may show up again in the context of Exam 2*. However, we will not directly ask you questions about Exam 1 material, however you might need this information to answer and/or understand the questions.

For example, we will not test you directly on ***loops*** but you will need to study and understand loops if you wish to iterate through all the elements in a **dictionary**, for instance.

# Strings

- Strings are immutable
- Strings and printing
  - print(the_string)
  - print("hi " + "there")
  - print("hi", str2)
- Different kinds of quotations, and mixing quotations
- Order matters - collection of characters

# Strings

- Each character is assigned an index, starting at index zero (0)
  - X[i]
  - X[0] - first character
  - Positive and negative indices:  str[-1] - last character
  - Getting one thing out of a collection
- Length of a string - len(str) - number of characters in the string
- Slicing - get multiple contiguous items out of the collection
  - Give it a start index and an end index
  - S[start:stop:step]  -- start position (incl), end position (excl.), the increment
  - Reversing a string:  my_string[: : -1]

# Strings: Slicing

```
• str1[start:stop]        # from start (incl.) to stop (excl.)
                            (through stop-1)

• str1[start:]            # from start (incl.) through the end of
                            the collection

• str1[:stop]             # from the beginning through to stop (excl.)
                            (through stop-1)

• str1[:]                 # a copy of the whole array

• str1[start:stop:step]   # from start (incl.) to stop (excl.)
                            (through stop-1), keeping every step^th item
```

# Strings

- Functions
  - "hi" in my_string - contains
  - str.lower()
  - str.upper()
  - str.startswith("hello")
  - str.endswith("world")
  - Note: return new value

  - str.replace(str1, str2)
  - str.strip()

str.count(str1) - returns total # of occurrences

str.find(str1) - returns lowest index

# Lists

- Collecting things together under one name
- Syntax: [ ] and items are comma separated
  - my_list = ['apple', 'banana']
- Mutable!
- Can mix types
- len(my_list)

# Lists

- Empty list: my_list = [ ]
- Adding:
  - my_list.append('watermelon')
  - my_list.insert(1, 'orange') - insert the item at the specified index
- Remove:
  - my_list.remove('apple')
- Index
  - my_list.index('banana') - gives you the index location
- Other
  - my_list.sort()

# Tuples

- Collecting things together under one name (like a List)
- Syntax: ( ) and items are comma separated
  - my_tuple = (1, 2, 3, 4)
- Immutable!
- Can mix types
- len(my_tuple)

# Tuples

- Empty tuple: my_tuple = ( )
- Adding…?:
  - ~~my_tuple.append('watermelon')~~
  - Immutable so can't use such mutating methods!
- Can use indexing of course:
  - my_tuple.[0]
- Index
  - my_tuple.index('Yellow') – gives you the index location
- Other
  - my_tuple.count(22) – how many times that item appears

# Lists vs. Tuples

## Reasons to use LISTS

- For flexibility of handling data: adding, deleting, changing (after list is created)
  - Mutability of Lists

## Reasons to use TUPLES

- When using data that doesn't or shouldn't change
  - Immutability of Tuples
  - (Structure to loop through and access the data without modifying the data)
- Python can access tuples faster than lists *(better performance)*

***Tuples are immutable and therefore do not have as many methods as a list. A lot of those lists methods that we have seen involved mutating the values.***
***For tuples, we cannot append, remove, etc.***
***Otherwise, lists and tuples behave in pretty much the same way!***

# Traversing Lists and Tuples

- Use the for-loop
- for item in collection:
        do stuff with variable item

# Dictionaries

- Ordering – not really; Indexing – no indexing
- A non-sequence data type
- Syntax: {key : value} and key-value pairs are comma separated
  - d = {4: "San Francisco", 7: "Edinburgh"}
- Instead of indices we use the index ourselves (an int, a string, …)
- Adding:
  - the_dict[12] = "London" -- add a key (12) value ("London") pair
- Retrieving:
  - X = the_dict[4] -- provide the key, get back the value

# Dictionaries

## LIST

- Index to access members
- Indexes start with 0
- Indexes are consecutive ints
- To add a new thing:
  `list.append(something)`

## DICTIONARY

- Has keys to access members
- Each key must be unique
- Key can be:
  - Strings, ints, floats, booleans, tuples
  - (Not: lists, sets, dictionaries)
- To add a new thing: `d[key]=value`

# Dictionary functions

*copy() - creates a copy of the dictionary*

```
p = orders.copy()
```

*keys() - returns iterator(sequence) to the set of key values in the dictionary*

```
for person in orders.keys():
```

*values() - returns iterator to the set of values in the dictionary*

```
for burger in orders.values():
```

*items() - returns iterator to the set of <key-value> pairs in the dictionary*

```
for pair in orders.items():
```

# More on Dictionaries

*(Assuming we have a dictionary named:* **`d`**)

**`d[key]`**
Return the value of *d* with key *key*. Raises a `KeyError` if *key* is not in the dictionary.

**`len(d)`**
Return the number of items in the dictionary.

**`key in d`**
Return `True` if *d* has a key *key*, else `False`.

**`list(d)`**
Return a list of all the keys in the dictionary.

**`del d[key]`**
Remove `d[key]` from *d*. Raises a `KeyError` if *key* is not in the dictionary.

**`d.popitem()`**
Remove and return a `(key, value)` pair from the dictionary. Pairs are returned in LIFO (most recent) order.

**`d.pop`**`(key[, default])` *`# default is optional`*
If *key* is in the dictionary, remove it and return its value, else return *default*. If *default* is not given and *key* is not in the dictionary, a `KeyError` is raised.

**`d.get`**`(key[, default])` *`# default is optional`*
Return the value for *key* if *key* is in the dictionary, else *default*. If *default* is not given, it defaults to `None`, so that this method never raises a `KeyError`.

```
instructors = {"001": "Lina", "002": "Jasmine", "003": "Kai"}

print("Lina" in instructors.values())
```

# Looping - Lists and Dictionaries

**List:**

Loop through a list by the **items** -

```
for my_item in my_list:
  Do stuff with my_item
```

Loop through a list by **index** -

```
for i in range(len(my_list)):
  my_item = my_list[i]
  Do stuff with my_item
```

**Dict:**

Loop through a dictionary -

```
for my_key in my_dictionary:
  Do stuff with my_key
  Do stuff with my_dictionary[my_key]
```

Can also loop through -

- d.keys()
- d.values()
- d.items()

```
for key, val in my_dictionary.items():
    # do stuff with key
    # do stuff with val
    ...
```

# Looping - Dictionaries

```
orders = {'Sofiya':'cheese burger', 'Jacob':'bbq burger', 'Kat':'mushroom burger', 'Xinyu':'cheese burger'}
# one way to print out the burgers
for person in orders: # looping through the keys
    # print(person) # person is the key
    print(orders[person]) # print the value that is stored at person
for person in orders.keys(): # another way to loop through keys, no different than the loop above
    print(orders[person])
print('a second way to access values')
for burger in orders.values():
    print(burger)
print('a third way to access values')
for pair in orders.items(): # Each pair is a tuple - (person, value)
    print(pair[1]) # the second item in the tuple
# Let's make a new dict to store the prices/costs
costs = {} # keys are people, the values are costs
for person in orders:
    costs[person] = float(input("how much does "+ person + ", owe? "))
print(costs)
```

# Nested Data Structures

- Nested Lists
  - nested_list = [[], []]

```python
# Iterating through the nested list
for sublist in matrix:
    for item in sublist:
        print(item)
```

- Nested Tuples
  - nested_tuple = ((), ())

- Nested Dictionaries
  - nested_dictionary = {{}, {}}

```python
# Nested Dictionary Example
school = {
    "classrooms": {
        "classroom1": {
            "capacity": 30,
            "students": ["Alice", "Bob", "Charlie"]
        },
        "classroom2": {
            "capacity": 25,
            "students": ["Dave", "Eve", "Frank"]
        }
    },
    "teachers": {
        "teacher1": {
            "name": "Ms. Johnson",
            "subject": "Math"
        },
        "teacher2": {
            "name": "Mr. Smith",
            "subject": "Science"
        }
    }
}
```

# Regular Expressions (RegEx)

- `[]` = brackets, matches a single character contained within
- `[x-z]` = hyphen, denotes a range of characters
- `a.b` = dot, matches any single character in that location
- `[^abc]` = caret, matches a single character that is not a, b, or c
- `c?` = preceding character (c) 0 or 1 times
- `c+` = preceding character (c) 1 or MORE times
- `c*` = preceding character (c) 0 or MORE times
- `a|b` = vertical bar, means or

# Regular Expressions (RegEx)

- `^` = matches the beginning of a string
- `$` = matches the end of a string
- `{m}` = exactly m copies of the previous RE
- `{m,n}` = from m to n copies of the previous RE
- `\b` = word boundary matching RE at beginning or end of a word
- `\d` = matches any decimal digit
- `\D` = matches any character that is not a decimal digit
- `\s` = matches any whitespace character
- `\S` = matches any character that is not a whitespace character
- `\w` = matches any word character
- `\W` = matches any character that is not a word character

# Q&A

I’m happy to address any questions you have about the exam!

# Reminder: CS Laptop Loaner Program

- This course requires students to have a **laptop**
- I realize that not everybody might have one (nor necessarily need one for their desired major / path…)
- If you do not have a laptop for any reason… ***not to worry!***
- The CS department's Systems staff has a notebook / laptop loaner program and will be able to loan you a notebook / laptop computer for the duration of the semester if you don't have one or if you cannot afford one.
  - Also available if your laptop is broken and under repair, we can arrange for you to receive a loaner laptop for a week or two until your own laptop is fixed

Interested? Link: https://www.cs.virginia.edu/wiki/doku.php?id=cs_laptop_loaner

*I am happy to be your sponsor. Please let me know.*