NYU – Tandon School of Engineering
Brooklyn, NY 11201

# CS-UY 1114 / Python

Final Exam – 16 December 2021

Name: ______________________________________________

NetID(first part of email):________________________________

- Duration: 2 hours and 0 minutes
- **DO NOT WRITE ON THE BACK OF ANY PAGE!**
- Do not separate any page.
- **Please do not use pencil**, if you must, write darkly, these pages will be scanned
- If you write an answer other than in the space provided, please indicate, in the space provided, where we can find that answer
- This is a closed book exam, no calculators are allowed
- You can expect that the user inputs the appropriate values (int/float/etc where required).
- Comments are not required
- Anyone found cheating on this exam will receive a zero for the exam.
- Anyone who is found writing after time has been called will receive a zero for this exam.
- Do not open this test booklet until you are instructed to do so.
- If you have a question please ask the proctor of the exam!
- You may use ONLY the following Python constructs and functions!

| | | |
|---|---|---|
| (all math operators) | if,elif,else | for |
| (all conditional operators) | while | ord |
| math.* (all in math module) | int | chr |
| random.* (all in random module) | print | input |
| str.* (all string methods) | len | str |
| list.* (all list methods) | try | except |
| dictionary.* (all dict methods) | open | class |
| read, readline and readlines | in | close |
| sys.* | return | |

- For reference, the point distribution for each question is below:

Question 1 - 15 points
Question 2 - 10 points
Question 3 - 8 points
Question 4 - 15 points
Question 5 - 25 points
Question 6 - 27 points

Name ________________________________________ Net ID: ___________________

1. **(15 Points) Provide the value of the variable "var" (use python form to show the appropriate data type, e.g. "str" , [list], etc). Note**: following code snippets will not produce errors.

| Question | Value of var after code runs |
|---|---|
| `var = 0`<br>`for i in range (2,5,3):`<br>`    for j in range(i,0,-1):`<br>`        var +=j` | |
| `var = []`<br>`ls = [1,2,3]`<br>`for i in ls:`<br>`    ls.pop()`<br>`    var.append( ls[:] )` | |
| `ls=[5, 10, 15, 20]`<br>`var = ls`<br>`ls.pop()`<br>`var[0]+=1` | |
| `quote = "a wse man can see the mssng lttrs"`<br>`message = quote[10:17] + " " + quote[-5:]`<br>`var = message.split(" ")` | |
| `area_codes = {"NY": "212",`<br>`              "NJ": "201",`<br>`              "DC": "202"}`<br><br>`del area_codes["DC"]`<br>`var = area_codes` | |

Name ________________________________________ Net ID: ___________________

2. (**10 Points)** Evaluate the code below and write its output.

```
def make_alphanum_codes(chars, num_list):
    i = 10
    for num in num_list[::-1]:
        for pos in range(len(chars)):
            code = chars[pos] + ',' + str(num)
            print(i, ':', code, sep='') #sep is empty str
            i - 1

n_list = [1, 2, 3]
alphas = "abc"

make_alphanum_codes(alphas, n_list)
```

Output

(use this area for scrap paper, do NOT remove any pages from this booklet, put your answers above)

Name ________________________________________ Net ID: ___________________

3. **(8 Points)** Fill in the blanks in the following table with the appropriate number system equivalents:

| **Decimal** | **Binary** | **Hexadecimal** |
|---|---|---|
| 25 | *(1 pt)* | *(1 pt)* |
| *(1 pt)* | 0001 0001 | *(1 pt)* |
| *(1 pt)* | *(1 pt)* | 2a |
| 120 | *(1 pt)* | *(1 pt)* |

(use this area for scrap paper, do NOT remove any pages from this booklet, put your answers above)

Name ________________________________________ Net ID: ___________________

**4.** **(15 Points)** Write a class named Airplane that represents each airplane in an airport. Each airplane has the following **attributes**:

- make (make of plain. Example: “Boeing”)
- model (model number. Example: 707)
- capacity (seating capacity. Example: 400)
- pilot (name of pilot)

The Airplane class should have an **initializer** (constructor) that accepts the airplane’s make, model, capacity, and pilot as arguments. The class should also implement the **`__str__()`** method that returns a string describing airplanes in this format:

```
Airplane description:              Airplane description:
Aircraft: Boeing 707               Aircraft: Airbus A380
Capacity: 400 seats                Capacity: 853 seats
Pilot: William Edward Boeing       Pilot: Chesley Burnett
```

a) Airplane class implementation:

b) Write code (you do not need to define any functions) to instantiate two airplane class objects.

Name ________________________________________ Net ID: ___________________

5. **(25 points)** A popular movie site provides a data feed with pairings of movie stars with the movies in which the star has appeared. This data has been processed to create a list of tuples where each tuple contains a movie star's name and the movie in which the star has appeared. An example of the data in this structure is shown below.

```
pairings = [
        ("Cate Blanchett", "Babel"),
        ("Johnny Depp", "Edward Scissorhands"),
        ("Jack Nicholson", "One Flew Over the Cuckoo's Nest"),
        ("Angela Bassett", "Waiting to Exhale"),
        ("Jack Nicholson", "Batman"),
        ("Jack Nicholson", "The Departed"),
        ("Cate Blanchett", "The Lord of the Rings: The Two Towers"),
        ("Angela Bassett", "Black Panther"),
        ("Brad Pitt", "Fight Club"),
        ("Jodie Foster", "The Silence of the Lambs"),
        ("Edward Norton", "American History X")
]
```

You can assume the following about the code that you implement:

- the list of movie star/movie title pairings will exist and will not be empty
- no main function is needed

a) First, we'd like to be able to search this list of pairings for actors that appear in a given movie (the cast). Define a function named **`get_cast()`** that accepts two parameters: the pairings list as described above, and a string with the name of a movie. The function will return a list of the names (i.e. a list of strings) of the actors that appeared in that movie. If the movie title doesn't appear in the list of pairings, return an empty list.

Function implementation:

b) In order to more easily work with the data, we would like to convert it to a dictionary. Define a function named **`convert`**`()` that accepts a single parameter: a list of pairings as described above (i.e. a list of tuples where each tuple contains a movie star's name and the movie in which the star has appeared). This function will return a dictionary where each movie star name that appears in the list is a key in a dictionary. The value associated with each key is a list of strings of the movie titles in which the star has appeared. (For example, `"Angela Bassett"` would be associated with `"Waiting to Exhale"` and `"Black Panther"`; `"Jodie Foster"` would be associated with `"The Silence of the Lambs"`, et cetera)

Function implementation:

c) Define a function named **`get_average`**`()` that accepts one parameter: a dictionary with movie star names (a string) and the movies in which the star has appeared (a list of strings) as produced from the **`convert`**`()` function above . Calculate and return the average number of movies each star appears in. You can calculate this by finding the total number of movies and dividing by the total number of actors. You do not have to worry about duplication of movie titles. In the above example, there are 11 movies and 7 stars so the average would be `11/7`.

Function implementation:

Name ________________________________________ Net ID: ___________________

6. **(27 points)** We will be creating a class that simulates a round of *mad lib*. A mad lib is a word game where you are given a paragraph such as this one:

```
This is a [N] of a [AJ] [N] that I will complete [AV] now.
Don't [V] it isn't [AJ] ok?
With [N] and [N] to you,
good night, [N]
```

This paragraph is missing a series of **nouns** (represented by **`[N]`**), **verbs** (represented by **`[V]`**), **adjectives** (represented by **`[AJ]`**), and **adverbs** (represented by **`[AV]`**). Your goal as a player is to fill these missing words in with any random noun, verb, adjective, or adverb.

For example, the mad lib below could be filled-in in the following way:

```
This is a fence of a delicious chair that I will complete quickly now.
Don't slice it isn't benign ok?
With fence and chair to you,
good night, cart
```

---

Define a class, **`MadLib`**, that accepts one parameter (`filepath`, a string) when instantiating.

The **`MadLib`** class will have a single instance attribute, **`contents`**, which will store the lines in the file pointed to by `filepath` **as elements of a list** (i.e. each line will be an element). No further processing is necessary.

To do this, the **`MadLib`** class must **attempt to safely open the file** to extract its contents. If anything goes wrong while attempting to open the file, `contents` will remain an empty string.

**`MadLib`** objects will have a single method associated with them: **`populate_madlib`**`()`. **`populate_madlib`**`()` will accept a single parameter, `word_bank`, a **`WordBank`** object.

**You can assume that the `WordBank` class is already defined and imported into your file**, and will have the following methods:

- **`get_verb`**`()`: Returns a string containing a random verb.
- **`get_noun`**`()`: Returns a string containing a random adjective.
- **`get_adjv`**`()`: Returns a string containing a random adjective.
- **`get_advb`**`()`: Returns a string containing a random adverb.

**`WordBank`** objects don't require any arguments in order to be properly instantiated.

**`populate_madlib`**`()` must use these methods to replace the four word tokens (**`[N]`**, **`[V]`**, **`[AV]`**, and **`[AJ]`**) with words from the **`WordBank`** object that was passed into it. After **`populate_madlib`**`()` is called, the **`contents`** attribute will contain the completed mad lib (i.e. the tokens are replaced by the appropriate words) instead of the version with the tokens.

You may assume that all tokens (**`[N]`**, **`[V]`**, **`[AV]`**, and **`[AJ]`**) will always be uppercase, and that they will **always** have a space character before and after them..

Name ________________________________________ Net ID: ___________________

Add functionality to your **`MadLib`** class so that, when passed into the **`print`**`()` function, it will print the current contents of the **`MadLib`** object's `contents` attribute **as they would appear in the txt file**.

To finish your file, define a **`main`**`()` function to show the **`MadLib`** class in action.

The **`main`**`()` function must extract the name of the `txt` file that will be used to create the **`MadLib`** object from the following command line/Terminal command:

*Mac / Linux*:

```
python3 madlib.py madlib.txt
```

*Windows*:

```
py madlib.py madlib.txt
```

The **`main`**`()` must then create a **`MadLib`** object, make a call to **`populate_madlib`**`()`, and pass the **`MadLib`** object into a **`print`**`()` function call. You can assume that the user will always enter a valid command into the Terminal (i.e. the correct number of arguments, no typos, etc.).

The output on your console, provided everything was properly implemented and went well, could look like this:

```
This is a chair of a benign fence that I will complete slovenly now.
Don't dry it isn't ostentatious ok?
With cart and chair to you,
good night, chair
```

Name ____________________________________________ Net ID: ____________________

Problem 5 Implementation:

```
from wordbank import WordBank
```

Name ________________________________________ Net ID: __________________

(Additional space and/or scrap, DO NOT SEPARATE ANY PAGES FROM THIS BOOKLET)

Name ________________________________________ Net ID: __________________

(Additional space and/or scrap, DO NOT SEPARATE ANY PAGES FROM THIS BOOKLET)