Week 10

COMP10001

Let’s chat: how can you improve your coding skills between project 1 and project 2?

Let’s chat: how can you improve your coding skills between project 1 and project 2?

- Commenting
  - Do your comments make your code more human-readable?
- Style
- Approach
- Debugging

Let’s chat: how can you improve your coding skills between project 1 and project 2?

- Commenting
  - Do your comments make your code more human-readable?
  - Do your docstrings describe function inputs, outputs and purpose?
- Style
- Approach
- Debugging

Let’s chat: how can you improve your coding skills between project 1 and project 2?

- Commenting
  - Do your comments make your code more human-readable?
  - Do your docstrings describe function inputs, outputs and purpose?
- Style
  - Are your variable names descriptive?
  - Do you use global constants to avoid magic numbers?
  - Does your code comply with PEP8? Use the filter!
- Approach
- Debugging

Let’s chat: how can you improve your coding skills between project 1 and project 2?

- Commenting
  - Do your comments make your code more human-readable?
  - Do your docstrings describe function inputs, outputs and purpose?
- Style
  - Are your variable names descriptive?
  - Do you use global constants to avoid magic numbers?
  - Does your code comply with PEP8? Use the filter!
- Approach
  - Have you use helper functions? (Not nested) Good helper functions improve readability, help with debugging and make your code more reusable!
- Debugging

Let’s chat: how can you improve your coding skills between project 1 and project 2?

- Commenting
  - Do your comments make your code more human-readable?
  - Do your docstrings describe function inputs, outputs and purpose?
- Style
  - Are your variable names descriptive?
  - Do you use global constants to avoid magic numbers?
  - Does your code comply with PEP8? Use the filter!
- Approach
  - Have you used helper functions? (Not nested) Good helper functions improve readability, help with debugging and make your code more reusable!
- Debugging
  - Do you use debugging strategies or do you find yourself getting stressed?

What is an “exception”?

What is an “exception”?

- Exception ~ run-time error
  - An event that disrupts the normal flow of the program’s instructions

What is an “exception”?

- Exception ~ run-time error
  - An event that disrupts the normal flow of the program’s instructions
  - The program stops running

What is an “exception”?

- Exception ~ run-time error
  - An event that disrupts the normal flow of the program’s instructions
  - The program stops running
  - AttributeError, IndexError, KeyError, NameError, TypeError, ValueError, FileNotFoundError, ZeroDivisionError

How can we use try, except and final to handle exceptions?

How can we use try, except and final to handle exceptions?

```
try:
    # code block where an exception might occur
except ExceptionType:
    # code block to handle the exception
finally:
    # code block that will always execute, regardless of
    # whether an exception was raised or not
```

How can we use try, except and final to handle exceptions?

```
try:
    # code block where an exception might occur
except ExceptionType:
    # code block to handle the exception
finally:
    # code block that will always execute, regardless of
    # whether an exception was raised or not
```

- If an exception occurs in the “try” block, the “except” block is run to handle the exception
- There can be multiple except statements after a single try

Write a function second_line(filename) that asks the user for the name of a file and then return the second line of the file. Use a try-except block to catch the file not found error and print the error message "Oh no, file not found". If exception is raised, return "ERROR" after printing the error message.

Write a function second_line(filename) that asks the user for the name of a file and then return the second line of the file. Use a try-except block to catch the file not found error and print the error message "Oh no, file not found". If exception is raised, return "ERROR" after printing the error message.

```
ERROR_MESSAGE = "Oh no, file not found"

def second_line(filename):
    try:
        with open(filename, 'r') as file:
            file.readline()
            return file.readline()
    except FileNotFoundError:
        print(ERROR_MESSAGE)
        return "ERROR"
```

What is an iterator? What are some helpful methods from the itertools library?

What is an iterator? What are some helpful methods from the itertools library?

- An iterator keeps track of the traversal of a container

What is an iterator? What are some helpful methods from the itertools library?

- An iterator keeps track of the traversal of a container
  - e.g. loops use iterators to keep track of iteration through a list

What is an iterator? What are some helpful methods from the itertools library?

- An iterator keeps track of the traversal of a container
  - e.g. loops use iterators to keep track of iteration through a list
- next(<iterator>)
  - Progress to the next item in the iterator
  - Raises a StopIteration exception if the end is reached

What is an iterator? What are some helpful methods from the itertools library?

- An iterator keeps track of the traversal of a container
  - e.g. loops use iterators to keep track of iteration through a list
- next(<iterator>)
  - Progress to the next item in the iterator
  - Raises a StopIteration exception if the end is reached
- Unlike containers (lists, sets, …), iterators can be infinite in length

The itertools library!

The itertools library!

- cycle - iterator to cycle through the items in a container, in a loop
- product - combine two containers into one tuple, with each item in one container combined with each item in the other
- combinations - a sequence of every possible combination of elements in a container
- permutations - like combinations, but including different orderings
- groupby - group elements of a container together in particular categories

What output does the following code print?

```
import itertools
beatboxer = itertools.cycle(['boots', 'and', 'cats', 'and'])

for count in range(39):
    print(next(beatboxer))
```

What output does the following code print?

```
import itertools

names = ['Amy', 'Alex', 'Bob']
animals = ['Cat', 'Dog']

print(list(itertools.product(names, animals)))
print(list(itertools.combinations(names, 2)))
print(list(itertools.permutations(names, 2)))
```

What output does the following code print?

```
import itertools

words = ['Cracker', 'Apple', 'Echidna', 'Egg', 'Aha', 'EmotionalDamage']

def first_char(word):
    return word[0]

words_group = itertools.groupby(sorted(words), first_char)
for key, group in words_group:
    print(key, list(group))
```

What output does the following code print?

```
import itertools

words = ['Cracker', 'Apple', 'Echidna', 'Egg', 'Aha', 'EmotionalDamage']

def first_char(word):
    return word[0]

words_group = itertools.groupby(sorted(words), first_char)
for key, group in words_group:
    print(key, list(group))
```

```
A ['Aha', 'Apple']
C ['Cracker']
E ['Echidna', 'Egg', 'EmotionalDamage']
```

If we don't sort the `words` list before doing `groupby`, then the output is

```
C ['Cracker']
A ['Apple']
E ['Echidna', 'Egg']
A ['Aha']
E ['EmotionalDamage']
```

Write a function which takes two strings as input and uses an `itertools` iterator to find whether the first word is an anagram of the second word. This might not be a very efficient way to find an anagram but it will help us work with iterators! `anagram('astronomer', 'moonstarer')` should return `True`

Write a function which takes two strings as input and uses an `itertools` iterator to find whether the first word is an anagram of the second word. This might not be a very efficient way to find an anagram but it will help us work with iterators! `anagram('astronomer', 'moonstarer')` should return `True`

```
from itertools import permutations

def anagram(word1, word2):
    for ordering in permutations(word1, len(word1)):
        if "".join(ordering) == word2:
            return True
    return False
```

Revision!

Write a function which takes a lowercase string as input and prints the frequency of each vowel in the string using a dictionary. `vowel_counts('i␣love␣python')` should print:

```
i 1
e 1
o 2
```

Revision!

Write a function which takes two lists of integers and returns the average of the numbers which they both have in common. `in_common_average([1, 2, 3, 4, 5], [0, 2, 4, 6])` should return `3.0`