COMP10001

2023 Sem 1

Foundations of Computing

Tutorial/Workshop

Week 10

Today's Tutorial

1. Project 1 feedback
2. exceptions
3. iterator

Project 1 feedback

- Helper functions!
  - Break up long functions
  - Not nested
- Commenting
  - Need docstring for each function, and some comments throughout
- Variable names
  - Descriptive, not just "a"
- Overly complicated
  - Subjective
  - Try utilise what we learn

Exceptions

Run-time errors are sometimes called exceptions, which is an event that disrupts the normal flow of the program's instructions.

By default, exceptions cause a program to **exit immediately**.

Common exceptions include:
`AttributeError`, `IndexError`, `KeyError`, `NameError`, `TypeError`, `ValueError`, `FileNotFoundError` and `ZeroDivisionError`.

Try – except – finally

```
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 code in the **except** block with the corresponding ExceptionType will run (as opposed to crashing).

With or without an exception, the **finally** block will run

Problem (not exercise) 1

Problem 1

1. 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.

Problem 1 answer

1. 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"
```

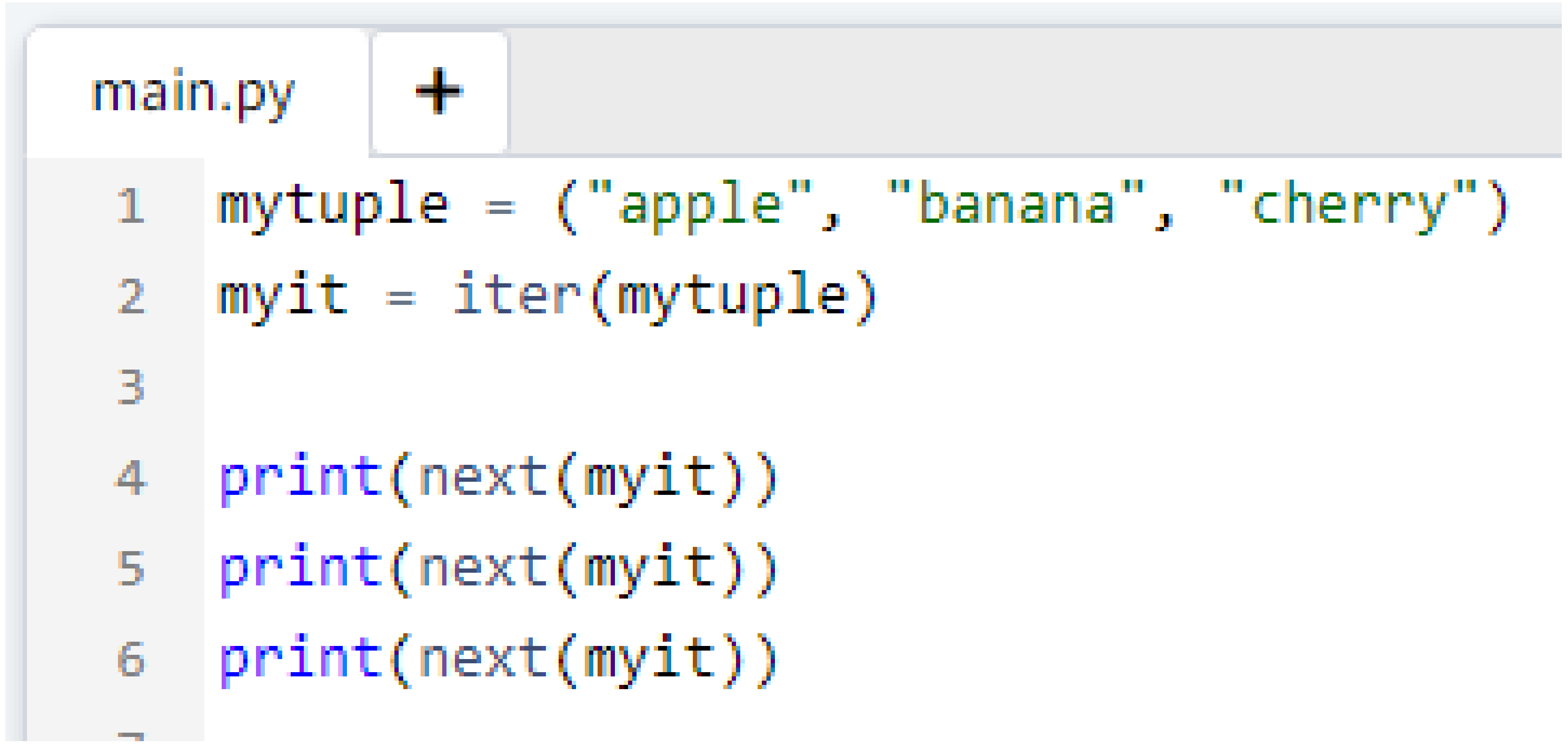
Iterators

An **object** that tracks iteration

An iterable object (tuple, list, dictionary, string) can have an iterator made of them

We can then use **next()** to iterate through

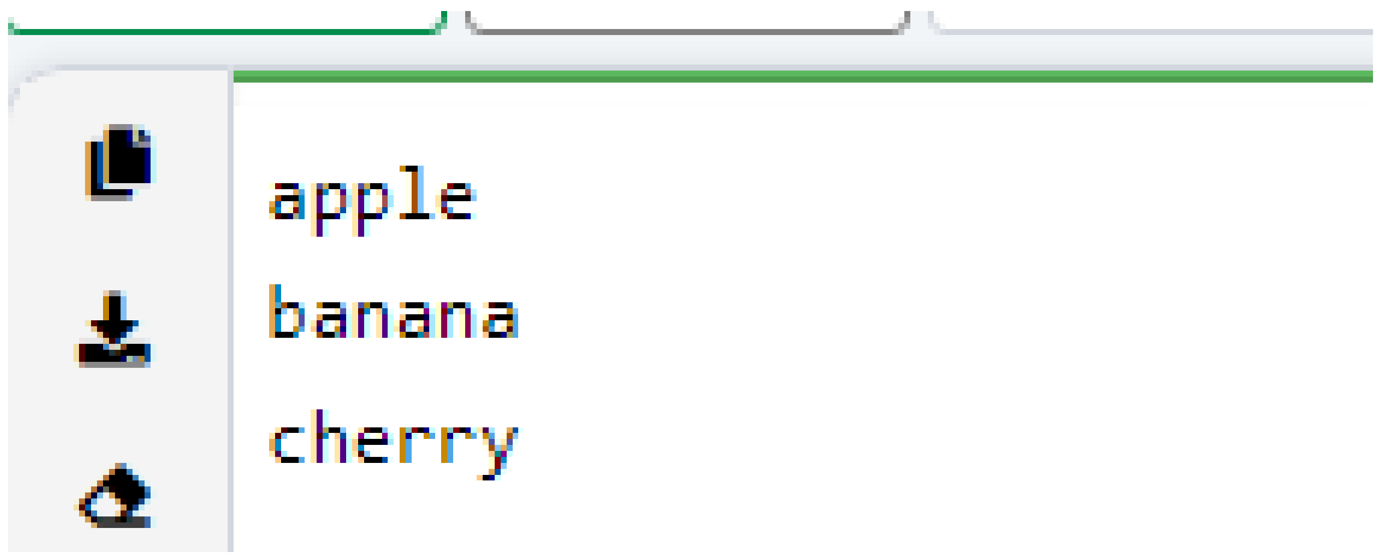
Iterators



main.py

```python
mytuple = ("apple", "banana", "cherry")
myit = iter(mytuple)

print(next(myit))
print(next(myit))
print(next(myit))
```



```
apple
banana
cherry
```

Itertools library

cycle(object) = iterator

can then cycle through, rather than just iterate

product(object, object) = tuple

the cartesian product of the objects

combinations(object, length (opt)) = list of tuples

all combinations of length

permutations(object) = list of tuples

all permutations

groupby(object, function (opt)) = iterator

ex: key_func = lambda x: x[0]

Exercises 1,2,3

Exercise 1 answer

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

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

A: *This code will print ten iterations of* `boots and cats and` *which will end with* `cats`*:*

```
boots
and
cats
and
boots
...
and
boots
and
cats
```

Try changing the `for` *loop to* `while True:` *(an infinite loop) to see this cycle print infinitely!*

Exercise 2 answer

2. 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)))
```

A:

```
[('Amy', 'Cat'), ('Amy', 'Dog'), ('Alex', 'Cat'), ('Alex', 'Dog'), ('Bob', 'Cat'), ('Bob', 'Dog')]
[('Amy', 'Alex'), ('Amy', 'Bob'), ('Alex', 'Bob')]
[('Amy', 'Alex'), ('Amy', 'Bob'), ('Alex', 'Amy'), ('Alex', 'Bob'), ('Bob', 'Amy'), ('Bob', 'Alex')]
```

Exercise 3 answer

3. What output does the following code print? What if we don't sort the `words` list before doing `groupby`?

```
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:

```
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']
```

WORKSHOP

Grok, problems from sheet, ask me questions :)

See you next week!