CS21: INTRODUCTION TO COMPUTER SCIENCE

Prof. Mathieson

Fall 2018

Swarthmore College

Outline Oct 24:

Sit somewhere new!

- Recap reading files
- String and List methods
- TDD: Top Down Design
  - **word_guesser.py**

Notes

•**Lab 6** due **Saturday** night
•Hand back stack diagram worksheet today
•Ninja session **tonight! 7-10pm**
•Office Hours **Friday 3-5pm**

CS after CS21

- Come talk to me if you’re interested in pursuing CS and have questions (31 vs. 35, etc)

- Even if you never take anything beyond CS21, takeaway: creativity under constraint

Screenshots and Videos: please email!

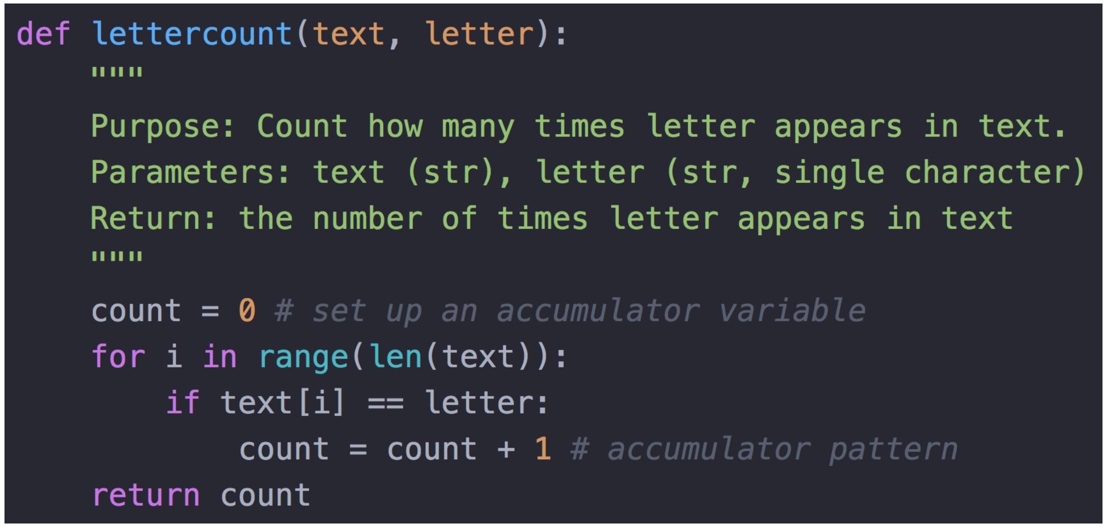
- Windows Videos: https://www.hongkiat.com/blog/win-screen-recording-softwares/
- Mac Videos: Quicktime
- Linux screenshot (camera icon on the bottom of screen)

Graphics on your own machine!
https://www.cs.swarthmore.edu/help/access.html
(need Xquartz (Mac) or Xming (Windows)

Recap reading files & Handout 4

Built-in vs. User-defined functions

- Both are ***functions***!
- User-defined example:



```python
def lettercount(text, letter):
    """
    Purpose: Count how many times letter appears in text.
    Parameters: text (str), letter (str, single character)
    Return: the number of times letter appears in text
    """
    count = 0 # set up an accumulator variable
    for i in range(len(text)):
        if text[i] == letter:
            count = count + 1 # accumulator pattern
    return count
```

- Built-in examples:
  - int(..)
  - print(..)
  - input(..)
  - random.choice(..)
  - random.randrange(..)
  - math.sqrt(..)

Built-in vs. User-defined functions

- Both are ***functions***!
- User-defined example:

```python
def lettercount(text, letter):
    """
    Purpose: Count how many times letter appears in text.
    Parameters: text (str), letter (str, single character)
    Return: the number of times letter appears in text
    """
    count = 0 # set up an accumulator variable
    for i in range(len(text)):
        if text[i] == letter:
            count = count + 1 # accumulator pattern
    return count
```

- Built-in examples:
  - int(..)
  - print(..)
  - input(..)
  - random.choice(..)
  - random.randrange(..)
  - math.sqrt(..)

Why are **random.choice(..)** and **random.randrange(..)** functions and not methods?

Built-in vs. User-defined functions

- Both are ***functions***!
- User-defined example:

```
def lettercount(text, letter):
    """
    Purpose: Count how many times letter appears in text.
    Parameters: text (str), letter (str, single character)
    Return: the number of times letter appears in text
    """
    count = 0 # set up an accumulator variable
    for i in range(len(text)):
        if text[i] == letter:
            count = count + 1 # accumulator pattern
    return count
```

- Built-in examples:
  - int(..)
  - print(..)
  - input(..)
  - random.choice(..)
  - random.randrange(..)
  - math.sqrt(..)

Why are **random.choice(..)** and **random.randrange(..)** functions and not methods?

Answer: **random** is a library/module, not a specific instance of a class.

Mini-quiz, discuss with a partner

```
c_file = open("colleges.txt",'r')
for line in c_file:
    tokens = line.split()
    name = tokens[0]
    enroll = int(tokens[1])
c_file.close()
```

1) What is the *type* of **c_file**? (conceptually)

2) What is the *type* of **line**?

3) What does **split** do?

4) What is the type of tokens?

Mini-quiz, discuss with a partner

```
c_file = open("colleges.txt",'r')
for line in c_file:
    tokens = line.split()
    name = tokens[0]
    enroll = int(tokens[1])
c_file.close()
```

1) What is the *type* of **c_file**? (conceptually)

file (technically **TextIOWrapper**)

2) What is the *type* of **line**?

3) What does **split** do?

4) What is the type of tokens?

Mini-quiz, discuss with a partner

```
c_file = open("colleges.txt",'r')
for line in c_file:
    tokens = line.split()
    name = tokens[0]
    enroll = int(tokens[1])
c_file.close()
```

1) What is the *type* of **c_file**? (conceptually)

file (technically **TextIOWrapper**)

2) What is the *type* of **line**?

string

3) What does **split** do?

4) What is the type of tokens?

Mini-quiz, discuss with a partner

```
c_file = open("colleges.txt",'r')
for line in c_file:
    tokens = line.split()
    name = tokens[0]
    enroll = int(tokens[1])
c_file.close()
```

1) What is the *type* of **c_file**? (conceptually)

file (technically **TextIOWrapper**)

2) What is the *type* of **line**?

string

3) What does **split** do?

Breaks up a string based on spaces.

4) What is the type of tokens?

Mini-quiz, discuss with a partner

```
c_file = open("colleges.txt",'r')
for line in c_file:
    tokens = line.split()
    name = tokens[0]
    enroll = int(tokens[1])
c_file.close()
```

1) What is the *type* of **c_file**? (conceptually)

file (technically **TextIOWrapper**)

2) What is the *type* of **line**?

string

3) What does **split** do?

Breaks up a string based on spaces.

4) What is the type of tokens?

list

Template for reading a file

1) Use a for loop to read the sequence of lines (recommended)

```
c_file = open("colleges.txt",'r')
for line in c_file:
    tokens = line.split()
    name = tokens[0]
    enroll = int(tokens[1])
c_file.close()
```

Template for reading a file

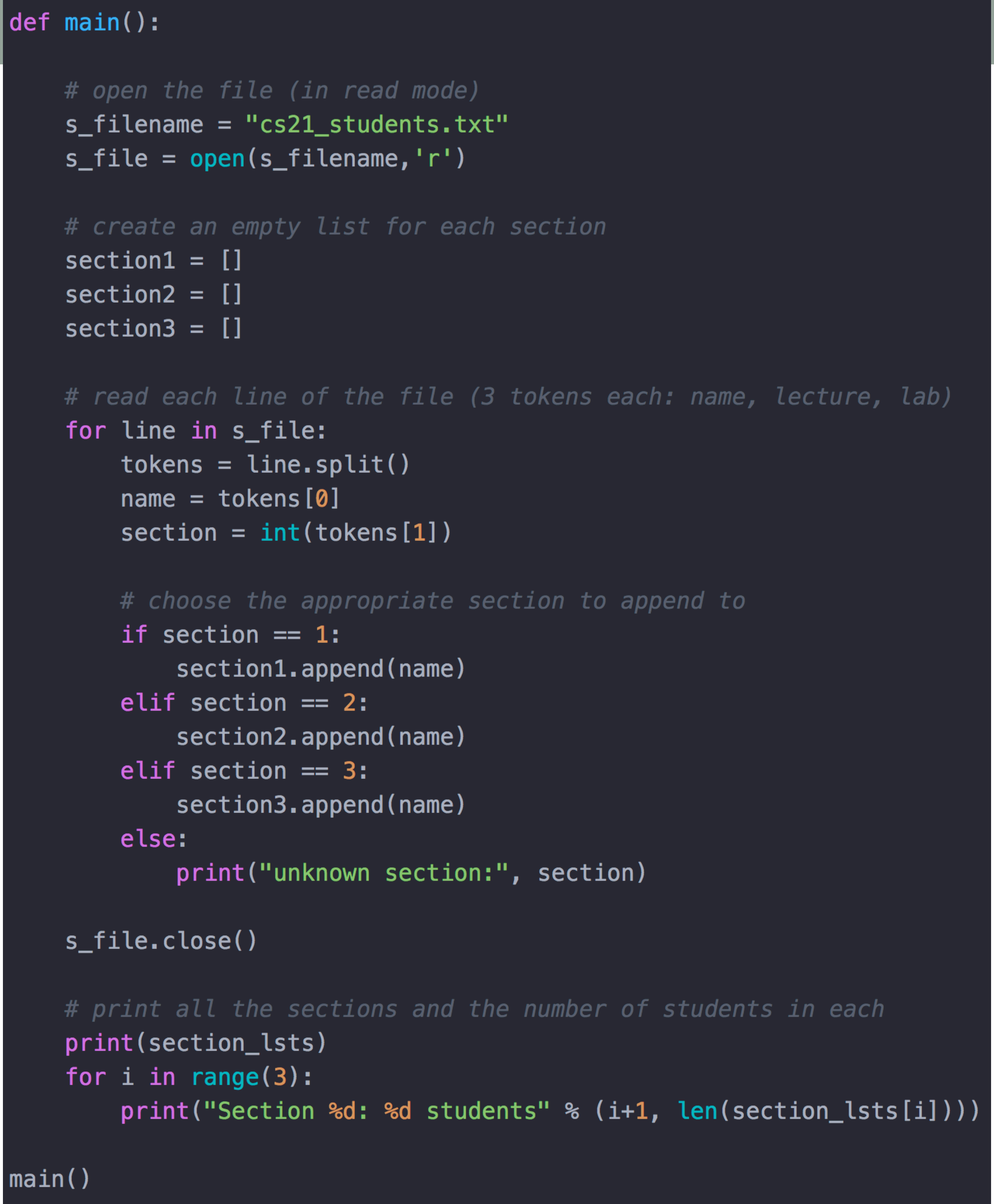
1) Use a for loop to read the sequence of lines (recommended)

```
c_file = open("colleges.txt",'r')
for line in c_file:
    tokens = line.split()
    name = tokens[0]
    enroll = int(tokens[1])
c_file.close()
```

2) Loop over the line indices (using readline() to get the next line)

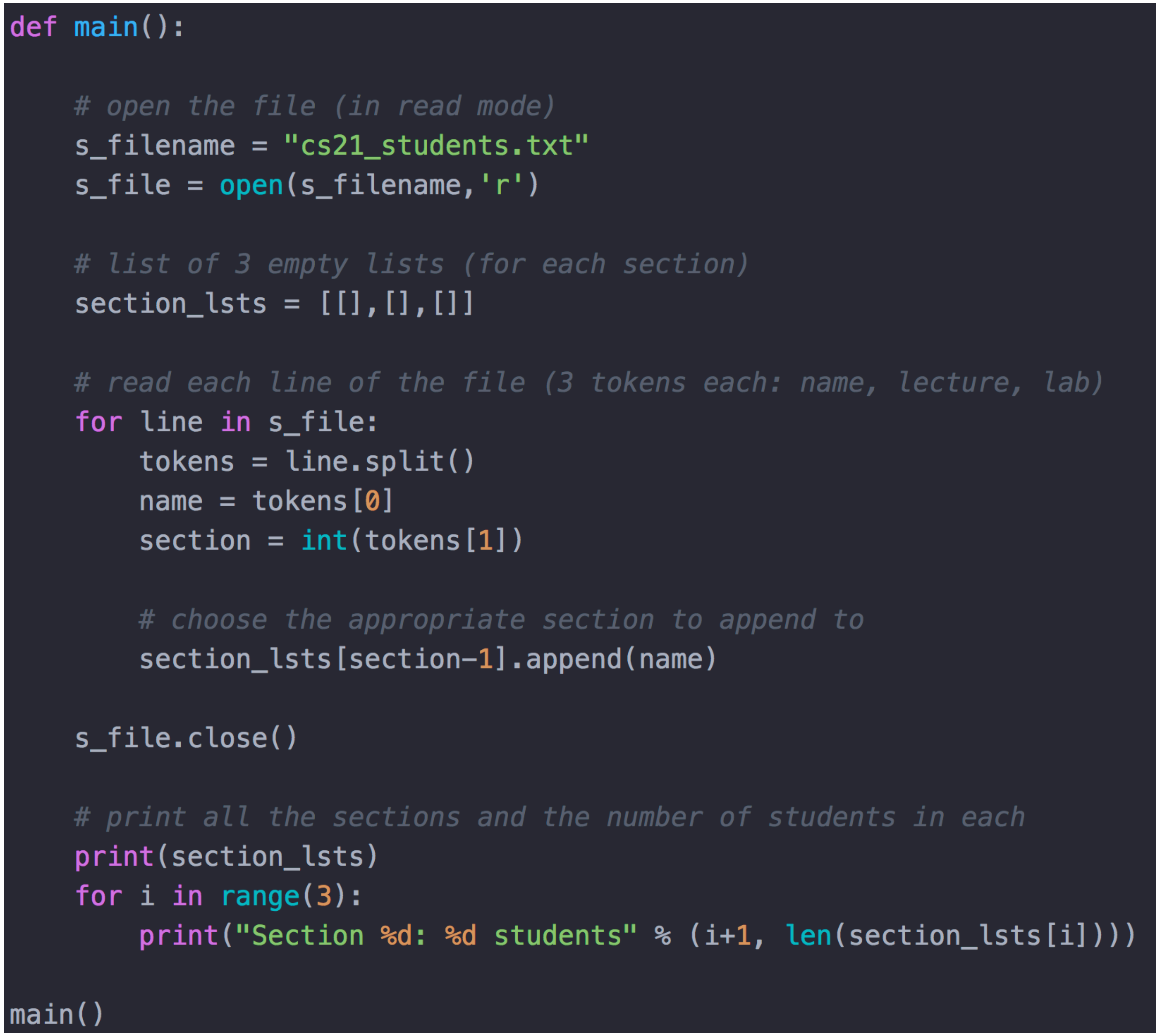
```
c_file = open("colleges.txt",'r')
for i in range(16):
    line = c_file.readline()
    tokens = line.split()
    name = tokens[0]
    enroll = int(tokens[1])
c_file.close()
```

students_file.py



```python
def main():

    # open the file (in read mode)
    s_filename = "cs21_students.txt"
    s_file = open(s_filename,'r')

    # create an empty list for each section
    section1 = []
    section2 = []
    section3 = []

    # read each line of the file (3 tokens each: name, lecture, lab)
    for line in s_file:
        tokens = line.split()
        name = tokens[0]
        section = int(tokens[1])

        # choose the appropriate section to append to
        if section == 1:
            section1.append(name)
        elif section == 2:
            section2.append(name)
        elif section == 3:
            section3.append(name)
        else:
            print("unknown section:", section)

    s_file.close()

    # print all the sections and the number of students in each
    print(section_lsts)
    for i in range(3):
        print("Section %d: %d students" % (i+1, len(section_lsts[i])))

main()
```

students_file.py



```python
def main():

    # open the file (in read mode)
    s_filename = "cs21_students.txt"
    s_file = open(s_filename,'r')

    # list of 3 empty lists (for each section)
    section_lsts = [[],[],[]]

    # read each line of the file (3 tokens each: name, lecture, lab)
    for line in s_file:
        tokens = line.split()
        name = tokens[0]
        section = int(tokens[1])

        # choose the appropriate section to append to
        section_lsts[section-1].append(name)

    s_file.close()

    # print all the sections and the number of students in each
    print(section_lsts)
    for i in range(3):
        print("Section %d: %d students" % (i+1, len(section_lsts[i])))

main()
```

Non-printing (“whitespace”) characters

\n	newline (appears at the end of each line in a file)
\t	tab
\s or **“ ”**	space

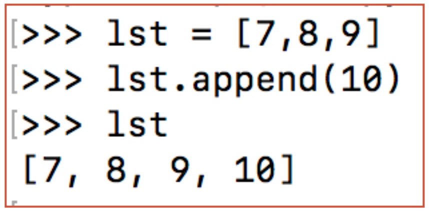
- Note: **<str>.strip()** removes leading and trailing whitespace

List and String Methods

Common List methods

Common List methods

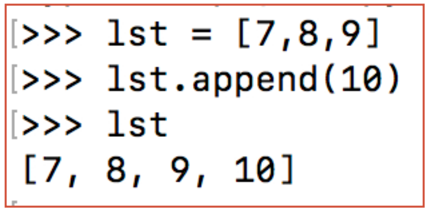
- Add a single element to a list: lst.append(item)



```
>>> lst = [7,8,9]
>>> lst.append(10)
>>> lst
[7, 8, 9, 10]
```

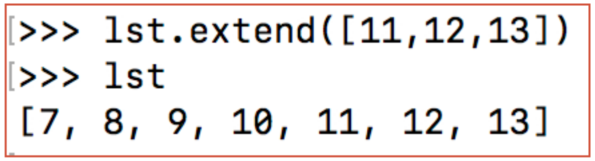
Common List methods

- Add a single element to a list: lst.append(item)



```
>>> lst = [7,8,9]
>>> lst.append(10)
>>> lst
[7, 8, 9, 10]
```

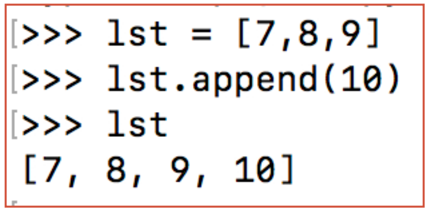
- Add a list to the end of a list: lst.extend(another_lst)



```
>>> lst.extend([11,12,13])
>>> lst
[7, 8, 9, 10, 11, 12, 13]
```

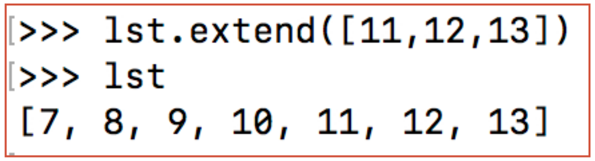
Common List methods

- Add a single element to a list: lst.append(item)



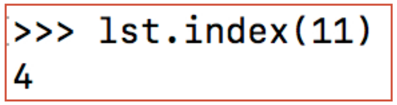
```
>>> lst = [7,8,9]
>>> lst.append(10)
>>> lst
[7, 8, 9, 10]
```

- Add a list to the end of a list: lst.extend(another_lst)



```
>>> lst.extend([11,12,13])
>>> lst
[7, 8, 9, 10, 11, 12, 13]
```

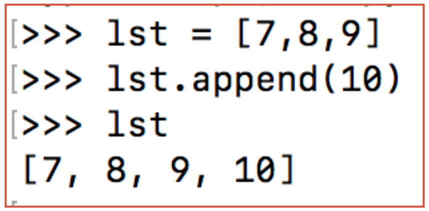
- Return the index of an element: idx = lst.index(item)



```
>>> lst.index(11)
4
```

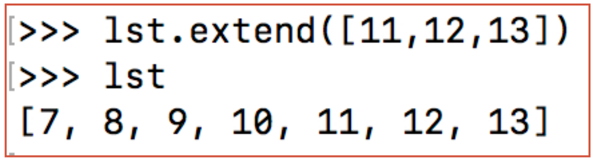
Common List methods

- Add a single element to a list: lst.append(item)



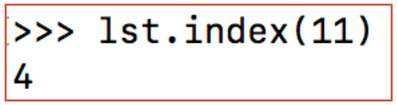
```
>>> lst = [7,8,9]
>>> lst.append(10)
>>> lst
[7, 8, 9, 10]
```

- Add a list to the end of a list: lst.extend(another_lst)



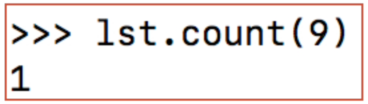
```
>>> lst.extend([11,12,13])
>>> lst
[7, 8, 9, 10, 11, 12, 13]
```

- Return the index of an element: idx = lst.index(item)



```
>>> lst.index(11)
4
```

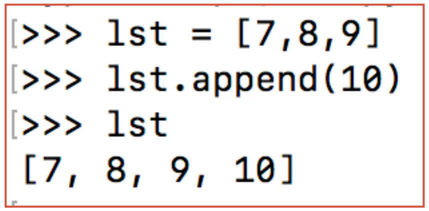
- Return the count of an element: c = lst.count(item)



```
>>> lst.count(9)
1
```

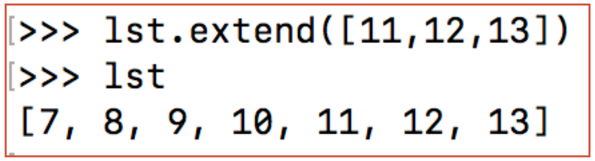
Common List methods

- Add a single element to a list: lst.append(item)



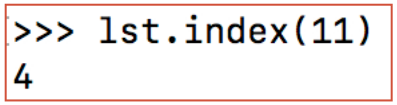
```
>>> lst = [7,8,9]
>>> lst.append(10)
>>> lst
[7, 8, 9, 10]
```

- Add a list to the end of a list: lst.extend(another_lst)



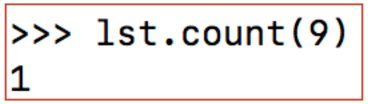
```
>>> lst.extend([11,12,13])
>>> lst
[7, 8, 9, 10, 11, 12, 13]
```

- Return the index of an element: idx = lst.index(item)



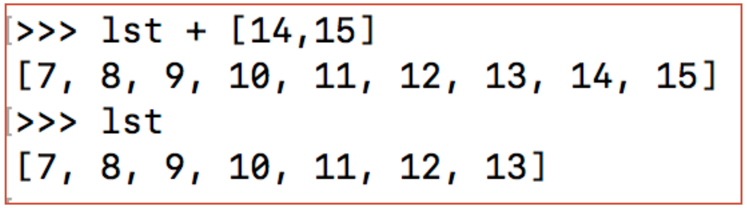
```
>>> lst.index(11)
4
```

- Return the count of an element: c = lst.count(item)



```
>>> lst.count(9)
1
```

- List concatenation (not a method): lst + another_lst



```
>>> lst + [14,15]
[7, 8, 9, 10, 11, 12, 13, 14, 15]
>>> lst
[7, 8, 9, 10, 11, 12, 13]
```

Common String Methods: they all return something!

- string.index(smaller_string)
- string.count(smaller_string)
- string.isalpha()
- string.lower()
- string.upper()
- string.split(smaller_string)
- string.strip()

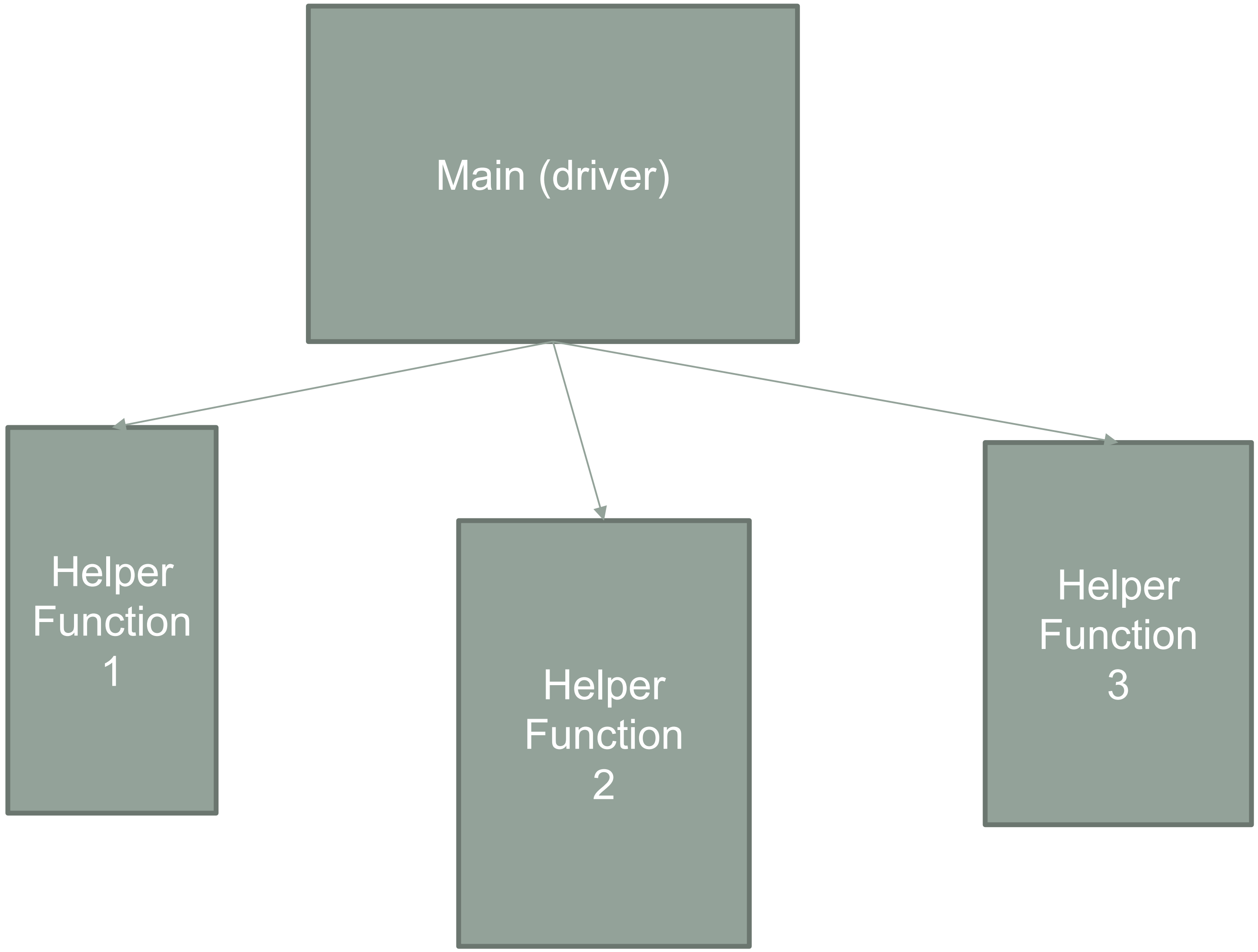
Common String Methods: they all return something!

string.index(smaller_string)	**int**
string.count(smaller_string)	**int**
string.isalpha()	**bool**
string.lower()	**string**
string.upper()	**string**
string.split(smaller_string)	**list**
string.strip()	**string**

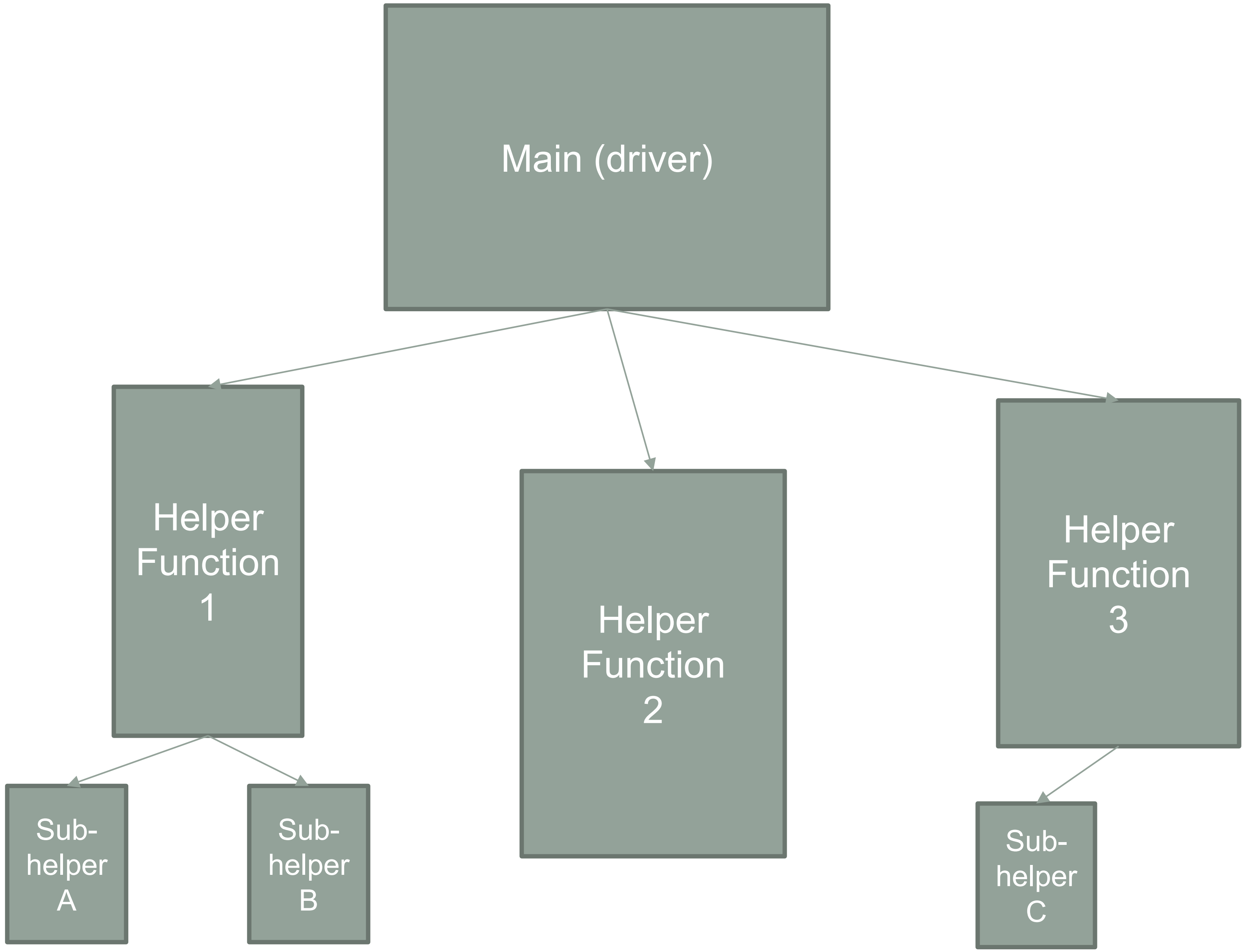
TDD
Top Down Design

Structure of main and “helper” functions

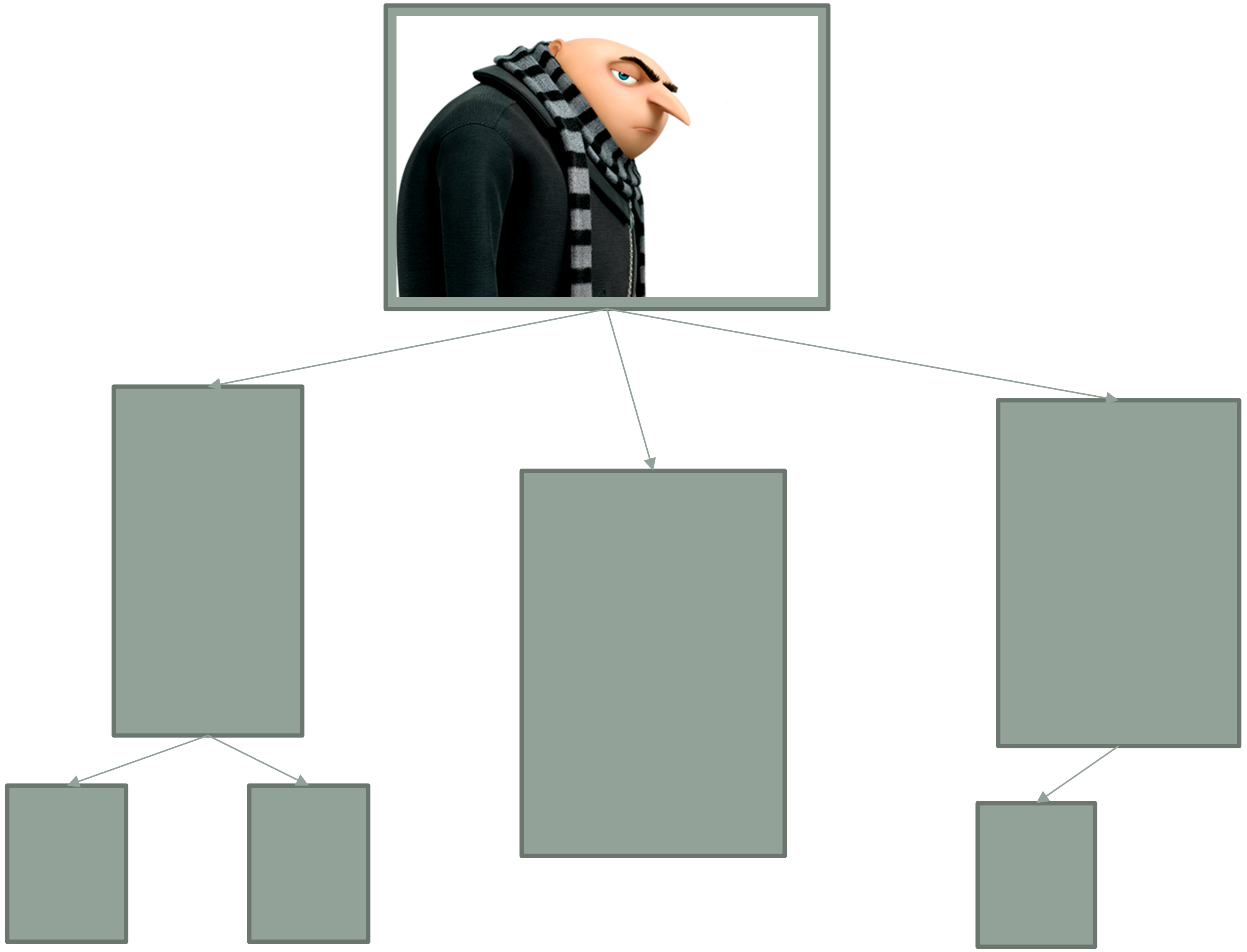
Structure of main and "helper" functions



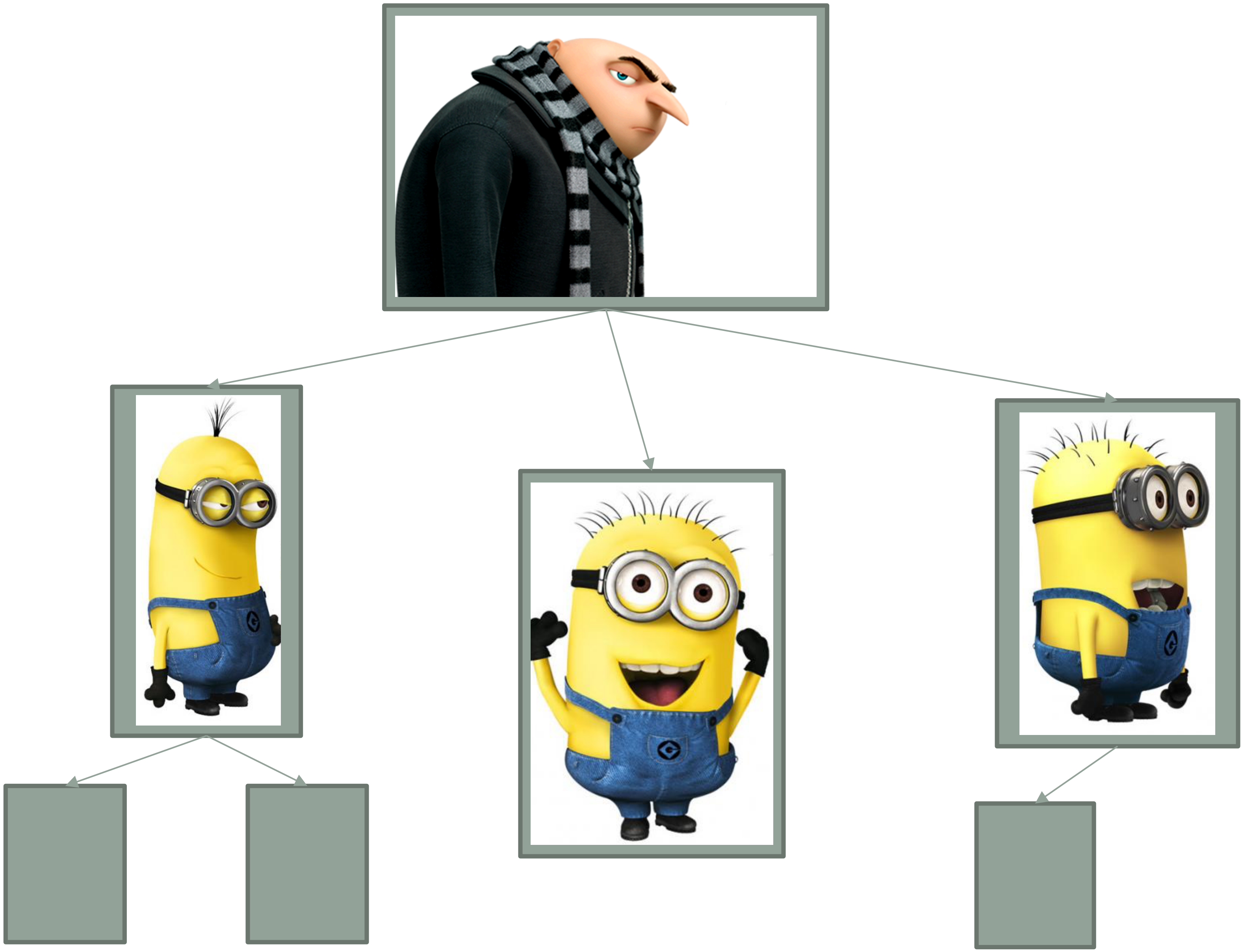
Structure of main and “helper” functions



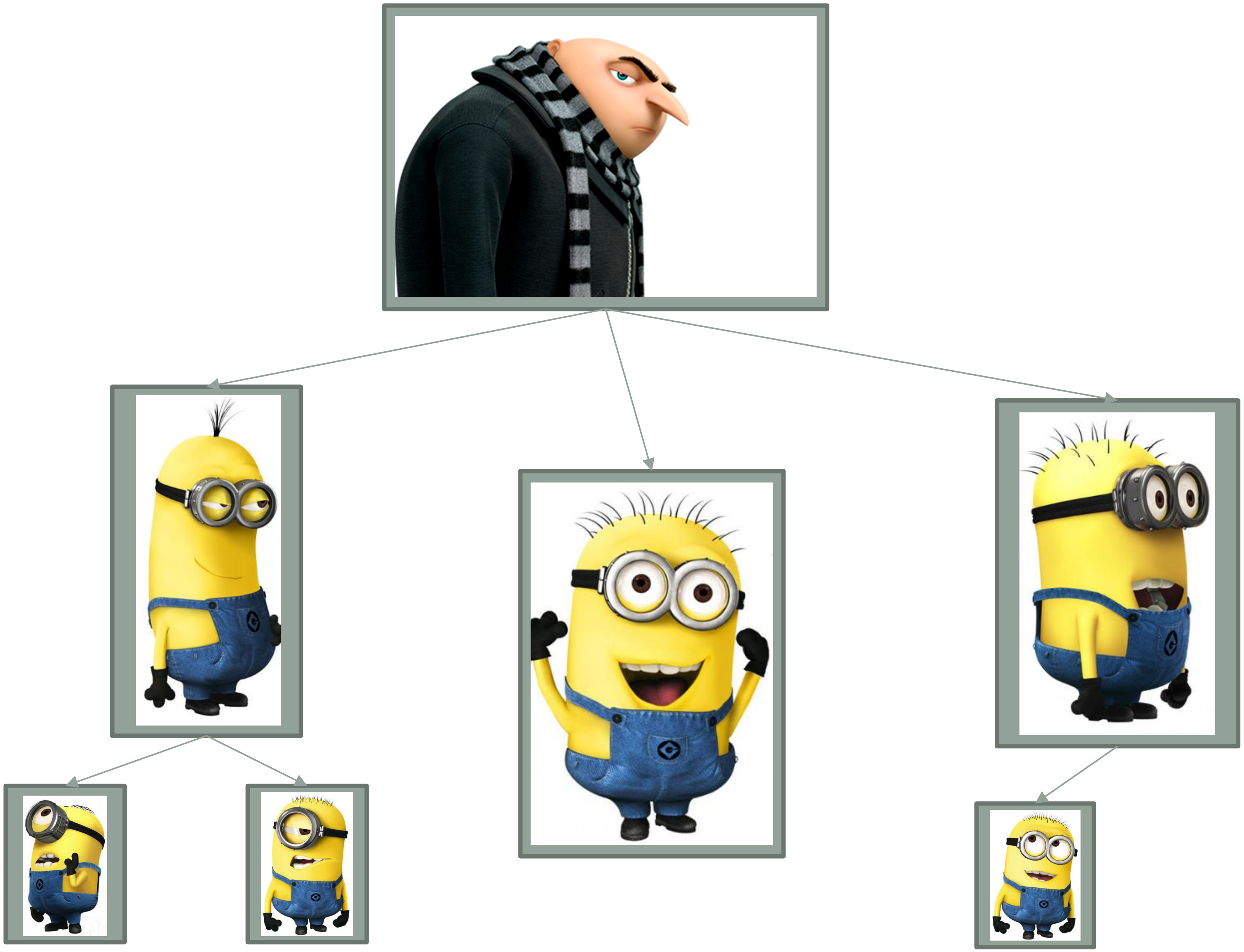
Structure of main and “helper” functions



Structure of main and “helper” functions



Structure of main and “helper” functions



Steps of TDD

Steps of TDD

1) Design a **high-level main function** that captures the basic idea of the program. Often this involves some initial variables, an outer loop, and some ending/output.

Steps of TDD

1) Design a **high-level main function** that captures the basic idea of the program. Often this involves some initial variables, an outer loop, and some ending/output.
2) As you're writing/designing main, think about which details can be **abstracted into small tasks**. Make names for these functions and write their signatures below main.

Steps of TDD

1) Design a **high-level main function** that captures the basic idea of the program. Often this involves some initial variables, an outer loop, and some ending/output.
2) As you're writing/designing main, think about which details can be **abstracted into small tasks**. Make names for these functions and write their signatures below main.
3) **"Stub" out the functions**. This means that they should work and return the correct type so that your code runs, but they don’t do the correct task yet. For example, if a function should return a list, you can return []. Or if it returns a boolean, you can return False.

Steps of TDD

1) Design a **high-level main function** that captures the basic idea of the program. Often this involves some initial variables, an outer loop, and some ending/output.

2) As you're writing/designing main, think about which details can be **abstracted into small tasks**. Make names for these functions and write their signatures below main.

3) **"Stub" out the functions**. This means that they should work and return the correct type so that your code runs, but they don’t do the correct task yet. For example, if a function should return a list, you can return []. Or if it returns a boolean, you can return False.

4) Iterate on your design until you have a working main and stubbed out functions. Then start **implementing** the functions, starting from the “bottom up”.

Reasons to use TDD

- Creates code that is easier to implement, debug, modify, and extend
- Avoids going off in the wrong direction (i.e. implementing functions that are not useful or don't serve the program)
- Creates code that is easier for you or someone else to read and understand later on