6. Programming basics part 2
LING 471

Assignment 2

- Posted on GitHub!
- You can start now!
  - We haven't covered everything, but you can certainly get files downloaded and configured
- Rubric and submission portal should already be available
- General task
  - Counting how many times each word occurs in a file
  - Printing something based on how many times a particular word occurred

Learning outcomes

- Describe the difference between a while-loop and a for-loop
- Write a while-loop and a for-loop
- Describe, create, and manipulate the dict data structure
- Use class methods
- Open and read files
- Open and write to files

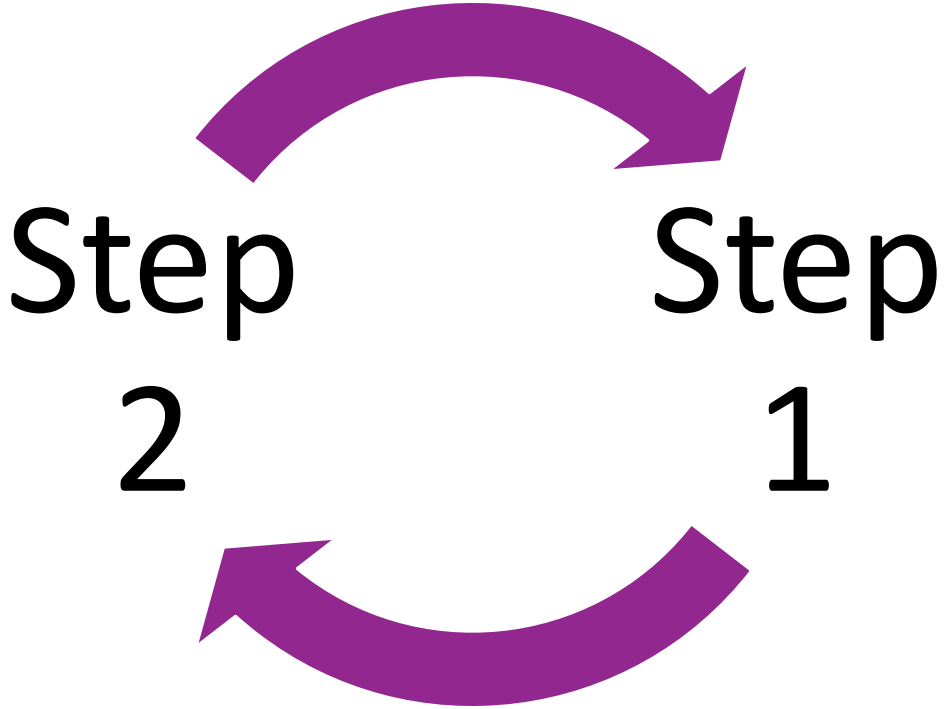
Iteration: `for` and `while` loops

Repeated tasks

- When giving instructions, it is common to give tasks that are repeated or involve repeated actions, e.g.,
  - Chop an onion (cut the onion until it is all chopped)
  - Serve each customer in line
  - While there is still dust on the floor, sweep up the dust
  - For every M&M in a bag, write down the color of the M&M
- Repeated tasks are also very common in programming!

Loops

- A loop is a set of instructions that are repeated 0 or more times
- Two general kinds of loops: `while` and `for`



Loops: `while`

- While some particular condition is true, perform a set of instructions
- `while COND:`
  `CODE`
  - `while` – keyword for while loop
  - `COND` – any Boolean expression;
  - `CODE` – the instructions to repeat

```
a = 1
while a <= 10:
    print(a)
    a = a + 1
```

Some important vocabulary and functions/keywords

VOCAB

- **Iterable**: some kind of object where it "makes sense" to be able to extract individual values
  - Real world: bag of marbles, list of instructions
  - Programming: lists, strings, ranges
- **Membership**: the presence of an object X in some sort of collection Y
  - Real world: Is Megan Skiendiel present in the list of students?
  - Programming: Is the number 3 present in the list of numbers?

FUNCTIONS

- `range(stop)` – creates an iterable that starts at 0 and stops at the integer immediately before **`stop`**
- `range(start, stop)` – creates an iterable that starts at **`start`** and stops at the integer immediately before **`stop`**
- `next(iterator)` – returns the next object from **`iterator`**; iterates sequentially when there is a natural order (e.g., for lists)

Checking membership: The `in` keyword

- We can check if an object is in a particular collection like a list or string by using the `in` keyword/binary operator
- X in Y
  - X – the object to look for
  - in – keyword
    Y – the object to check in
- Usually only want to use `in` with certain kinds of data structures like sets and dicts, sometimes strings, and almost never lists

```
a = [1, 2, 3]
3 in a # True
4 in a # False

s = 'hello'
'h' in s # True
'ha' in s # False
```

Loops: `for`

- For every item in some collection, perform a specific set of instructions
- for VAR in ITERABLE:
  CODE
  - for – keyword indicating for loop
  - VAR – the variable name for each object
  - in – keyword that checks membership
  - ITERABLE – object to iterate over
  - CODE – instructions to repeat

```
a = [1, 2, 3, 4]
for x in a:
    print(x)

# prints each element in a

for x in range(1, 5):
    print(x)
# prints each value from the
# specified range

for x in range(len(a)):
    a[x] = a[x] * 2
# a = [2, 4, 6, 8]
```

Loop guards

- Your computer is very happy to loop through a set of instructions forever
- We have to explicitly tell the computer when to stop iterating/looping
- For a `while` loop, the guard is the condition given in the statement
- For a `for` loop, the guard is running out of items to iterate over
  - Other languages specify guards for `for` loops differently
  - A `for` loop is often safer than a while loop because the guard is built-in

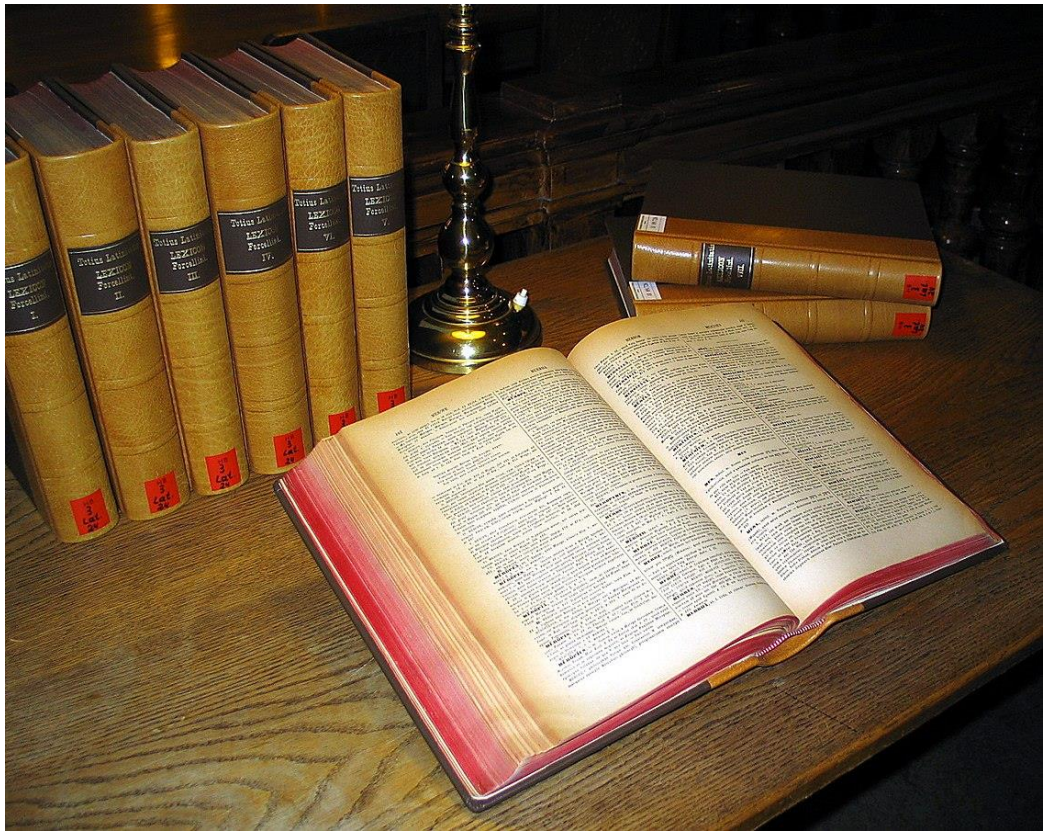
Programming activities

- Using a while loop, write code that will print the integers from 1 up to and including 10
- Create a list of integers from 1 up to and including 5 and assign it to a variable
  - Check if the number 10 is in it
- Using a for loop, write code that will print the integers from 1 up to and including 10
- Using a for loop, print out every character in the string 'ling471'
- Thinking activity: What would happen if you used “while True:” for a while loop? What about “while False:” and “for x in []”?
  - You can test your guesses to see!

Dicts

Real-world dictionaries

- Think about how a dictionary works in real life
- You have a pairing between a headword and a definition
- That is, you look up (or index!) a definition by the headword
- There are similar data structures in programming!



Python dicts

- In Python, the data structure that maps between something like a headword and a definition is called a **dict** or **dictionary**
- More technically, a dict maps between keys and values
- Keys – analogous to headword in dictionary
  - Can be any kind of immutable data
- Values – analogous to definition in dictionary
  - Can be any kind of data

Creating a dict

- Dictionaries can be created syntactically in Python
- {k_1:v_1, k_2:v_2, …}
  - { - curly brace indicating start of dictionary
  - k_1:v_1, k_2:v_2, … - a series of 0 or more key-value pairings
    - key (index) is on the LHS, value is on the RHS
  - } – curly brace indicating end of dictionary

```
food_types = {'pinto':'bean',
'banana':'fruit',
'rice':'grain'}

word_counts = {'the':10,
'brown':4, 'dog':2}

bad_dogs = {} # empty dict
```

Indexing a dict

- You can index into a dict (like looking something up)
  - Uses same indexing syntax as lists and strings
- You cannot slice a dict, though!
- If you index something that is not in the set of keys, you will get a KeyError

```
word_counts = {'the':10,
'brown':4, 'dog':2}

word_counts['the'] # 10

word_counts['brown'] # 4

word_counts[10] # KeyError
```

Adding to or modifying a dict

- Once you've created a dict, you can add or modify key-value pairs by combining indexing and assignment
- Hint: This kind of instructions will be **particularly** useful for assignment 2 ;)

```
word_counts = {'the':10,
'brown':4, 'dog':2}

word_counts['the'] = 11

word_counts['hello'] = 1
```

Checking for membership in a dict

- You can check if a key is in a dict using the `in` keyword
- This membership check is much faster than for a list or a string
  - Using `in` with a string or list must search through the object
  - With a dict (or set), the computer just needs to look something up

```
word_counts = {'the':10,
'brown':4, 'dog':2}

'dog' in word_counts # True

'pig' in word_counts # False
```

Programming activities

- Create an empty dict
  - Add the pair `'a':1` to it
  - Add the pair `1:'b'` to it
  - Check if 2 is in the dict
  - Check if `'a'` is in the dict
  - Increase the value of `'a'` to 2
- Using a for loop, modify an empty dict to have keys that are the integers 1 up to and including 4 and values that are the squares of the integers 1 up to and including 4 (that is, 1^2, 2^2, 3^2, 4^2)

Class methods; input/output (I/O)

Class methods

- Many objects in Python have what are called class methods
- These methods often manipulate, transform, or change that object
- You access them using the dot operator, `.` (aka, a period)
- The dot will separate the object and the method you are calling
  - For our general purposes, a method is basically a function or set of instructions
- You often need to look these up in the official Python documentation

String methods

- Any string in Python has a lot of useful methods you can use
- Remember that strings are immutable, so these methods return new strings

```
s = 'abcdefg’
s.upper() # 'ABCDEFG’

s = 'here are some words'
s.split() # ['here', 'are', 'some',
'words’]

s = '  there is extra space here '
s.strip() # 'there is extra space
here'

', '.join(['Smith', 'Janet',
'student']) # 'Smith, Janet,
student'
```

List methods

- There are also a lot of methods you can perform on a list
- Many (but not all) of these operations will **mutate** (change) the list

```
a = [1, 2, 3]
a.append(4) # a == [1, 2, 3, 4]

a = [3, 1, 2]
a.sort() # a == [1, 2, 3]

a = [1, 2, 4]
a.insert(2, 3) # a == [1, 2, 3, 4]

a = [1, 1, 1, 2, 1, 3, 1]
a.count(1) # 5
```

Reading in a file (input)

- Frequently, we want to be able to read in information from a file
  - If we don't we have to manually store all of the information in our Python code file
  - Imagine doing that for the entire *The Lord of the Rings* trilogy! (You'd have a long Python file…)
- Python allows us to read these files in in various ways

The `with` keyword

- When working with files, it is generally safest to use the `with` keyword
  - Otherwise, you have to manually remember when to tell Python to close the file stream (stop reading from the file)
- Creates a new context where an expression is stored in a variable
- with EXPR as VAR:
  CODE
  - with – keyword for with
  - EXPR – an expression to evaluate
  - as – second keyword (you just need to use it)
  - VAR – the variable to store the value of EXPR as
  - CODE – the code where you use VAR

Filestreams

- Python provides a function called open, which will open a filestream
- A filestream lets you
  - Read from an already created file
  - Write to a new file or an already created one
- A File object is created when you use open

Opening and reading a file

- We need to tell Python we want to read a file
  - Give **`open`** the **`'r'`** argument
- Once we've created our File object, we can tell Python to read the contents into a string or a list of strings for each line

```
with open('myfile.txt', 'r') as f:
    s = f.read()

# s contains the entire file as a
string

with open('myfile.txt', 'r') as f:
    s = f.readlines()

# s contains a list where each
element is a line from the file
```

Opening and writing to a file

- We need to tell Python we want to be able to write to a file
  - Give **`open`** the **`'w'`** argument
- Once we've created our File object, we can tell Python what to write out to the file
- ***CAUTION***: If you tell Python to write to a file that already exists, it will delete that file first!

```
with open('myfile.txt', 'w') as f:
    f.write('this is line 1\n')

# myfile.txt now contains the text
"this is line 1" with a newline
character at the end
```

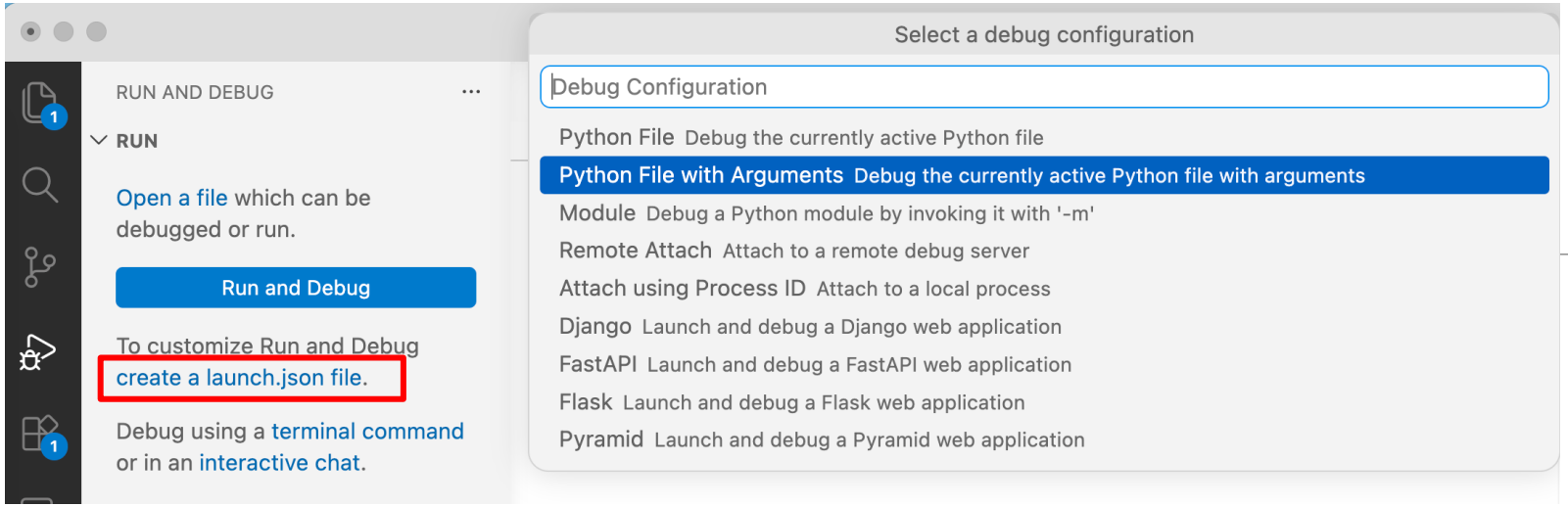
Escape characters

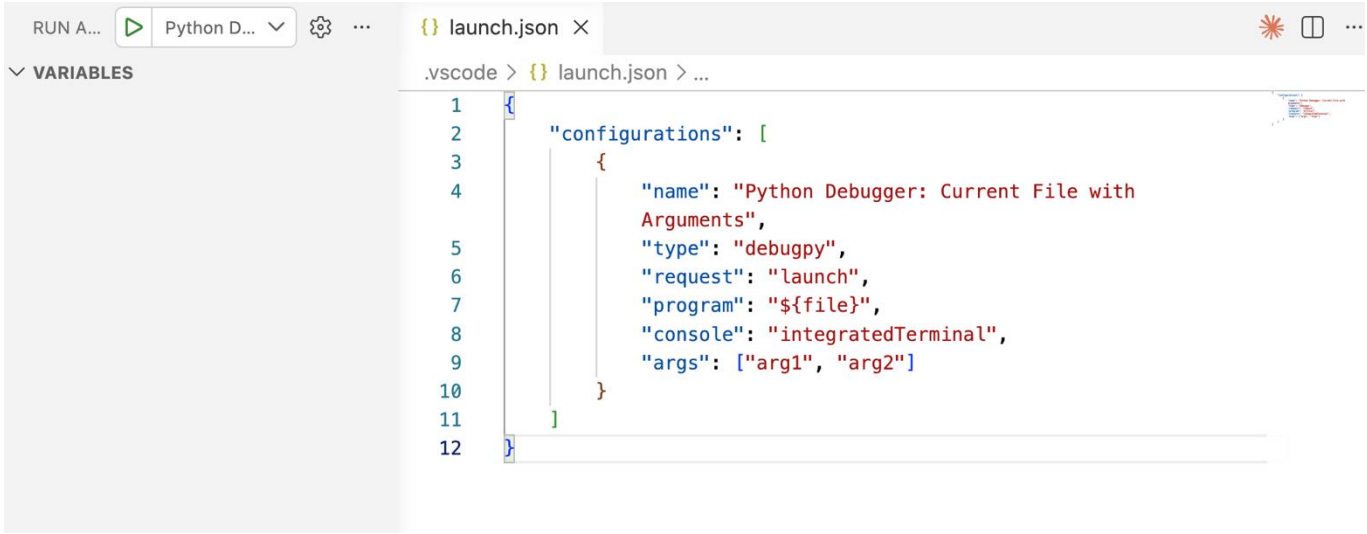
- Some characters are hard to type in strings
  - E.g., a line break, a tab character, a single quote mark, a double quote mark, etc.
- These characters can be typed with escape characters
- Escape characters are preceded by a backslash and then the character the interpreter should use

- Line break: \n
- Tab: \t
- Single quote: \'
- Double quote: \"
- Backslash: \\

A special list: sys.argv

- When you run a Python program, you can give it additional parameters/arguments to work with
- This can be useful if you want to tell the program to work on a specific file, e.g., `python myscript.py myfile.txt`
- You access these arguments from the `sys.argv` variable
- `sys.argv[0]` is the script name, and `sys.argv[1]` and on are the arguments passed in separated by whitespace, if any
- In the above example, `sys.argv[1]` would be 'myfile.txt'
- NB: need to have had the instruction `import sys` before trying to access `sys.argv`





- Run & Debug -> create a launch.json file -> Python Debugger -> Python File with Arguments
- Set multiple arguments at the line ”args”: [”arg1”, “arg2”, …]

Programming activity

- Create a txt file with all lower case text and save it somewhere you can find
- Read that txt file in with Python
  - Hint: you will need use the `read` method from a File object
- Change all the text to uppercase
  - Hint: there is a string method called `upper` that can do this
  - Write the txt file back out to a new file
  - Hint: you will need to use the `write` method from a File object