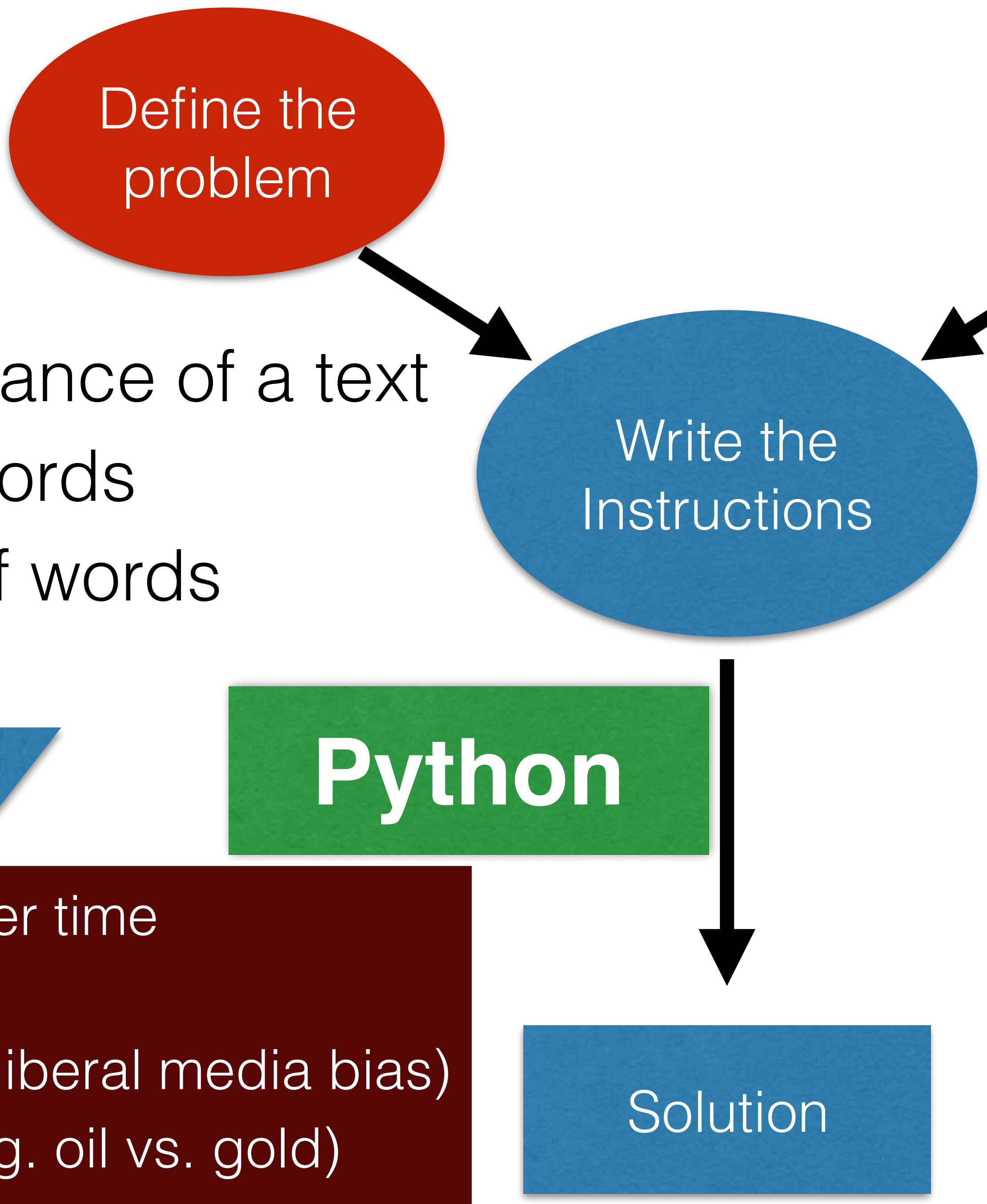
2.2 Loops Conditionals Reading Files

Grace Ngai

Much of this course and lots of the lecture notes were inspired by or derived from Brown University's CS931. Our thanks go to Prof Shriram Krishnamurthi and Hammurabi Mendes for their kind permission in allowing us to use their materials.

Textual Analysis



Build a concordance of a text

- *Locations* of words
- *Frequencies* of words

Word frequencies over time
Author of texts
Bias of authors (e.g. liberal media bias)
Worldwide trends (e.g. oil vs. gold)
…

Today

- Get some Statistics!
    - Read in a text file and create a list of words
    - Count the number of words in Tiger, Tiger Burning Bright (by William Blake)
    - Find the average length of a word in “Tiger”
    - Count the number of times the word “Tiger” is used in “Tiger”

Reminder from last week

- **`myString.split()`**
- **`split()`** returns a list of words in a string
  - Separated by whitespace
  - Can also use a specified delimiter
- In programming: we **call `split()` on** the string variable **`myString`**
- **`split()`** is what we call a **function** (more on that later)

Do Act 2-2, Task 1

```
  myString.split()
myString.split("\n")
```

string to "work on"

Empty, or a substring that will act as a *delimiter*

Working with Files

- If we want to read in large amounts of text (e.g. Legco records), we will have to read them in from files.
- Files on the computer have names like:
  - C:\Users\Grace\Desktop\ACT2-2\myFile.txt (Windows)
  - /Users/Grace/Desktop/ACT2-2/myFile.txt (Mac)
  - The file that we are working with will need to be stored in the same directory as our program, otherwise our program isn't going to be able to find it.
    - (There are other ways, but this is the easiest for now.)

Reading in a File

- Before we read in a file, we need to open it.

```
filename = “myFile.txt”
infile = open(filename, “r")
fileString = infile.read()
infile.close()
```

- The variable **`infile`** is called a *filehandle*. It represents the file inside the Python program.

- The string `"r"` after the name of the file tells Python that we are going to read the file.

- If the file does not exist, Python will give us an error.

Reading in a file

```
filename = “myFile.txt”
infile = open(filename, “r")
fileString = infile.read()
infile.close()
```

- The **`read()`** command tells Python to read in all the contents of **`infile`** at once.
    - The file is read in as one big string of characters and stored in the variable (**`fileString`** in this example)
    - Alphabetical symbols, numbers, spaces, newlines, ….
- The **`close()`** command does some “cleanup” after we’re done reading in the file.

Example

```
filename = “tiger.txt”
infile = open(filename, “r")
fileString = infile.read()
infile.close()
print(fileString)
```

```
Tiger, tiger, burning bright
In the forests of the night,
What immortal hand or eye
Could frame thy fearful symmetry?
In what distant deeps or skies
Burnt the fire of thine eyes?
On what wings dare he aspire?
What the hand dare seize the fire?
And what shoulder and what art
Could twist the sinews of thy heart?
And when thy heart began to beat,
What dread hand and what dread feet?
What the hammer? what the chain?
In what furnace was thy brain?
What the anvil? What dread grasp
Dare its deadly terrors clasp?
When the stars threw down their spears
And water'd heaven with their tears,
Did He smile His work to see?
Did He who made the lamb make thee?
Tiger, tiger, burning bright
In the forests of the night,
What immortal hand or eye
Dare frame thy fearful symmetry?
```

Do Activity 2-2, Task 2

Review: Sequential Operation

- Last week, we learned how Python runs programming statements *sequentially*
  - Programming statements are executed one after the other, in the order that they appear in the program.
  - Each statement waits for the previous one to finish before starting.

```
x = 5      # assign x
y = 6      # assign y
z = x+y    # assign z
print(“x is”, x)
print(“z is”, z)
x = 13     # update x
print(“x is now”, x)
print(“z is still”, z)
```

```
x is 5
z is 11
x is now 13
z is still 11
```

Repeating Things

- Consider the following program

```
x = [“Hong”, “Kong”, “Polytechnic”, “University”]
print(“Word 1 is”, x[0])
print(“Word 2 is”, x[1])
print(“Word 3 is”, x[2])
print(“Word 4 is”, x[3])
```

```
Word 1 is Hong
Word 2 is Kong
Word 3 is Polytechnic
Word 4 is University
```

- This is very repetitive

- Also prone to human error!

Repeating Things

- Python gives us an easy way of *iterating* over all the elements in a list:

```
x = [“Hong”, “Kong”, “Polytechnic”, “University”]
for item in x:
    print(“The word is”, item)
```

```
The word is Hong
The word is Kong
The word is Polytechnic
The word is University
```

Repeating Things

- Try the following in IDLE:

```
for i in [1, 2, 3, 4, 5]:
    print(i)
```

```
for j in [2, 4, 6, 8, 10, 12]:
    print(j)
```

```
for myVariable in [3, 7, 1, 2, 8, 2]:
    print(myVariable)
```

- What does the `for` … `in` … do?

Python Loops

- Repeating things in programming is called *looping*.
- Python loops have the following syntax:

```
for <variable> in <list>:
    <body>
```

- The body of the loop is a block of program statements.
- The statements in the loop body are **indented** to indicate the start and the end of the loop.
- Python will perform the statements in the **loop body** once for every item in the sequence.

Example of Loop

```
x = [“Hong”, “Kong”, “Polytechnic”, “University”]
num = 1
for word in x:
    print(“Word”, num, “is”, word)
    num = num + 1
```

```
Word 1 is Hong
Word 2 is Kong
Word 3 is Polytechnic
Word 4 is University
```

Note that the loop body is **indented (preceded by a tab)**
This indicates that these two program statements are “inside” the loop.

```
x = [“Hong”, “Kong”, “Polytechnic”, “University”]
num = 1
for word in x:
    print(“Word”, num, “is”, word)
    num = num + 1
```

Variable	Value

```
x = [“Hong”, “Kong”, “Polytechnic”, “University”]
num = 1
for word in x:
    print(“Word”, num, “is”, word)
    num = num + 1
```

Variable	Value
x	[“Hong”, “Kong”, “Polytechnic”, “University”]

```
x = [“Hong”, “Kong”, “Polytechnic”, “University”]
num = 1
for word in x:
    print(“Word”, num, “is”, word)
    num = num + 1
```

Variable	Value
x	[“Hong”, “Kong”, “Polytechnic”, “University”]
num	1

```
x = [“Hong”, “Kong”, “Polytechnic”, “University”]
num = 1
for word in x:
    print(“Word”, num, “is”, word)
    num = num + 1
```

Variable	Value
x	[“Hong”, “Kong”, “Polytechnic”, “University”]
num	1
word	“Hong”

```
x = [“Hong”, “Kong”, “Polytechnic”, “University”]
num = 1
for word in x:
    print(“Word”, num, “is”, word)
    num = num + 1
```

Variable	Value
x	[“Hong”, “Kong”, “Polytechnic”, “University”]
num	1
word	“Hong”

```
Word 1 is Hong
```

```
x = [“Hong”, “Kong”, “Polytechnic”, “University”]
num = 1
for word in x:
    print(“Word”, num, “is”, word)
    num = num + 1
```

Variable	Value
x	[“Hong”, “Kong”, “Polytechnic”, “University”]
num	2
word	“Hong”

```
Word 1 is Hong
```

```
x = [“Hong”, “Kong”, “Polytechnic”, “University”]
num = 1
for word in x:
    print(“Word”, num, “is”, word)
    num = num + 1
```

Variable	Value
x	[“Hong”, “Kong”, “Polytechnic”, “University”]
num	2
word	“Kong”

```
Word 1 is Hong
```

```
x = [“Hong”, “Kong”, “Polytechnic”, “University”]
num = 1
for word in x:
    print(“Word”, num, “is”, word)
    num = num + 1
```

Variable	Value
x	[“Hong”, “Kong”, “Polytechnic”, “University”]
num	2
word	“Kong”

```
Word 1 is Hong
Word 2 is Kong
```

```
x = [“Hong”, “Kong”, “Polytechnic”, “University”]
num = 1
for word in x:
    print(“Word”, num, “is”, word)
    num = num + 1
```

Variable	Value
x	[“Hong”, “Kong”, “Polytechnic”, “University”]
num	3
word	“Kong”

```
Word 1 is Hong
Word 2 is Kong
```

```
x = ["Hong", "Kong", "Polytechnic", "University"]
num = 1
for word in x:
    print("Word", num, "is", word)
    num = num + 1
```

Variable	Value
x	["Hong", "Kong", "Polytechnic", "University"]
num	3
word	"Polytechnic"

```
Word 1 is Hong
Word 2 is Kong
```

```
x = [“Hong”, “Kong”, “Polytechnic”, “University”]
num = 1
for word in x:
    print(“Word”, num, “is”, word)
    num = num + 1
```

Variable	Value
x	[“Hong”, “Kong”, “Polytechnic”, “University”]
num	3
word	“Polytechnic”

```
Word 1 is Hong
Word 2 is Kong
Word 3 is Polytechnic
```

```
x = [“Hong”, “Kong”, “Polytechnic”, “University”]
num = 1
for word in x:
    print(“Word”, num, “is”, word)
    num = num + 1
```

Variable	Value
x	[“Hong”, “Kong”, “Polytechnic”, “University”]
num	4
word	“Polytechnic”

```
Word 1 is Hong
Word 2 is Kong
Word 3 is Polytechnic
```

```
x = [“Hong”, “Kong”, “Polytechnic”, “University”]
num = 1
for word in x:
    print(“Word”, num, “is”, word)
    num = num + 1
```

Variable	Value
x	[“Hong”, “Kong”, “Polytechnic”, “University”]
num	4
word	“University”

```
Word 1 is Hong
Word 2 is Kong
Word 3 is Polytechnic
```

```
x = [“Hong”, “Kong”, “Polytechnic”, “University”]
num = 1
for word in x:
    print(“Word”, num, “is”, word)
    num = num + 1
```

Variable	Value
x	[“Hong”, “Kong”, “Polytechnic”, “University”]
num	4
word	“University”

```
Word 1 is Hong
Word 2 is Kong
Word 3 is Polytechnic
Word 4 is University
```

```
x = [“Hong”, “Kong”, “Polytechnic”, “University”]
num = 1
for word in x:
    print(“Word”, num, “is”, word)
    num = num + 1
```

Variable	Value
x	[“Hong”, “Kong”, “Polytechnic”, “University”]
num	5
word	“University”

```
Word 1 is Hong
Word 2 is Kong
Word 3 is Polytechnic
Word 4 is University
```

```
x = [“Hong”, “Kong”, “Polytechnic”, “University”]
num = 1
for word in x:
    print(“Word”, num, “is”, word)
    num = num + 1
```

Variable	Value
x	[“Hong”, “Kong”, “Polytechnic”, “University”]
num	5
word	“University”

```
Word 1 is Hong
Word 2 is Kong
Word 3 is Polytechnic
Word 4 is University
```

```
x = [“Hong”, “Kong”, “Polytechnic”, “University”]
num = 1
for word in x:
    print(“Word”, num, “is”, word)
    num = num + 1
```

- The variable **`word`** is called the *loop index*
- The list has the elements **`“Hong”, “Kong”, “Polytechnic”, “University”`**
- There is another variable, **`num`**
- Each time we go through the loop, we:
    - “pick up” the next element in the list
    - Execute the two programming statements in the loop body
    - i.e. the loop executes once for **`“Hong”`**, once for **`“Kong”`**, once for **`“Polytechnic”`**, once for **`“University”`**

Do Activity 2-2, Task 3

Making Decisions

- Revision: Python programs usually operate *sequentially* — i.e. each statement is executed in order, with one statement finishing before the next one is executed
- We have seen one exception: *loops* allow us to *repeat* certain programming statements for a given number of times.
- Conditional (*if*) Statements allow us to *skip* certain programming statements, subject to certain conditions.

Conditional Statements

- A conditional statement is also called an "`if`" statement

```
if <condition>:
    <body>
```

- <*condition*> is a Python expression that will give either "True" or "False" as an answer.

- <*body*> is a block of Python programming statements that are executed only if the answer of <*condition*> is True.

Conditional Statements

```
x = 3
if x % 2 == 0:
   print(x, “is even”)
```

- The first statement, `x = 3`, is executed.
- The next statement, the `if`, evaluates the result of `x%2` (the remainder when `x` is divided by 2), and checks if the result is equal to 0.
  - `x % 2` is equals to 3 % 2 which is equals to 1, which is not equals to 0
  - Therefore, the answer to the condition is False
- The `print` statement is therefore not executed.

Conditional Statements

```
x = 8
if x % 2 == 0:
   print(x, “is even”)
```

```
8 is even
```

- The first statement, `x = 8`, is executed.
- The next statement, the `if`, evaluates the result of `x%2` (the remainder when `x` is divided by 2), and checks if the result is equal to 0.
    - `x % 2` is equals to 0
    - Therefore, the answer to the condition is True
- The `print` statement is executed.

Comparing things

- The heart of an `if` statement is the condition
- A condition compares the value of two expressions

```
celsius > 30
```

```
x % 2 == 0
```

- The format of a condition is

<expr> <rel-op> <expr>

- *<rel-op>* stands for **relational operator**
- Python has six relational operators

Comparing Things

- Note the middle bit in the if statement

```
if x % 2 == 0 :
```

- This is a condition.
- A condition compares the values of two expressions

```
celsius > 30
```

```
x % 2 == 0
```

- The format of a condition is

<expr> <rel-op> <expr>

- *<rel-op>* stands for **relational operator**
- Python has six relational operators

Comparing Things

- Python has six relational operators — i.e. six ways in which we can compare things

Python	Mathematics	Meaning
`<`	$<$	Less than
`<=`	$\leq$	Less than or equal to
`==`	$=$	Equal to
`>=`	$\geq$	Greater than or equal to
`>`	$>$	Greater than
`!=`	$\neq$	Not equal to

Comparing Things

- The relational operators allow us to compare expressions.
- Python allows us to compare any two values, as long as they are of the same type
  - Numbers compare with numbers, strings with strings.

```
>>> 3 == 3
True
>>> 3 != 3
False
>>> 6 > 3
True
>>> 6 > 7
False
>>> 7/2 == 3.5
True
>>> "cat" == "cat"
True
>>> "cat" == "dog"
False
>>> len(“cat") > 2
True
```

Finish Activity 2-2