Booleans
Conditionals
Programming Patterns

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

Define the problem

Find the Data

Build a concordance of a text
- *Locations* of words
- *Frequencies* of words

Write the Instructions

Python

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

Solution

What we've learned thus far…

Data in Python

- Numeric (int, float)
  - 3, 4.5, 3.1415, 10, -1
- Textual (strings)
  - `“The big red fox”`, `‘CY Leung’`, `‘3.1415926’`
- Lists (sequences)
  - `[1, 3, 5, 7, 9]`, `[“a”, “panda”, “ocean”, “park”]`

Commands in Python

- Commands : also called “programming statements”
- Expressions: Statements that calculate a value
  - `3+5`, `“a”*3`, `“poly” + “u”`, `[1, 3] + [5]`
- Assignments: Statements that store a value into memory in the form of a *variable*
  - `myNum = 3`
  - `studentsInClass = [“1400001d”, “Henry Wong”, “15000001d”]`

Sequential Operation

- Programming statements in Python are executed ***sequentially***, one after the other, in the order that they are "seen" by the computer.
    - The computer doesn't "go back" and reevaluate programming statements *unless it is told to.*

```
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
```

Iterations

- Iterations are instructions to Python to ***repeat***, or ***iterate***, over certain ***blocks*** of programming code.

- Iterations in Python are also called **`for` loops**.

- The block of code that is to be repeated must be (1) indented to show that they are “under” the for loop, and (2) be in one contiguous block (i.e. we can’t put a programming statement that “doesn’t belong in the block” in the middle of the block.)

Iterations

```
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
```

```
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
```

Conditionals / Selections

- Conditional / selection statements are instructions to Python to ***selectively*** execute blocks of programming statements depending on the result of some ***condition***.

- Selection statements are also called **`if`** statements.

- The block of code that is to be executed (or not) must be (1) indented to show that they are “under” the for loop, and (2) be in one contiguous block (i.e. we can’t put a programming statement that “doesn’t belong in the block” in the middle of the block.)

Selection Statements

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

```

```

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

```
8 is even
```

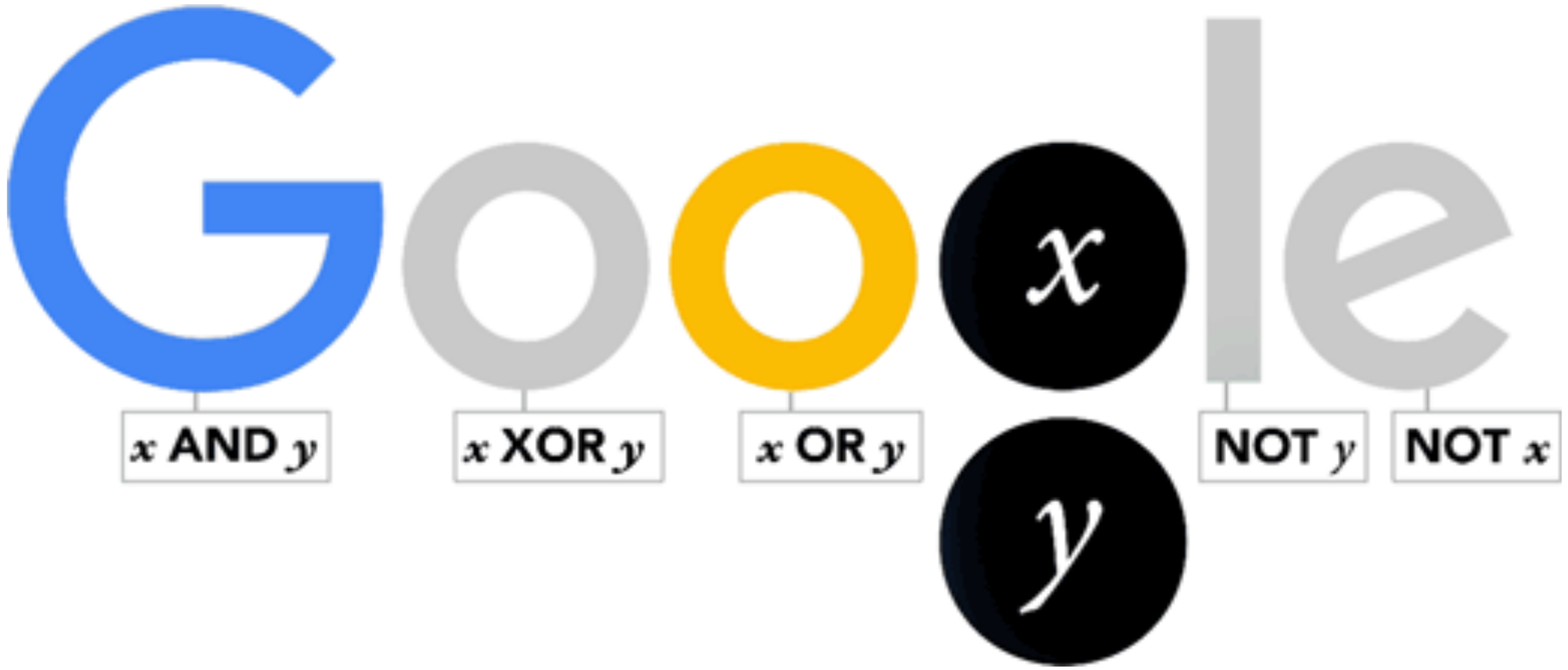
```
x = [1, 2, 3, 4, 5]
for item in x:
  if item % 2 == 0:
      print(item, “is even”)
```

```
2 is even
4 is even
```

This is a **`if`** inside a **`for`**!

Today

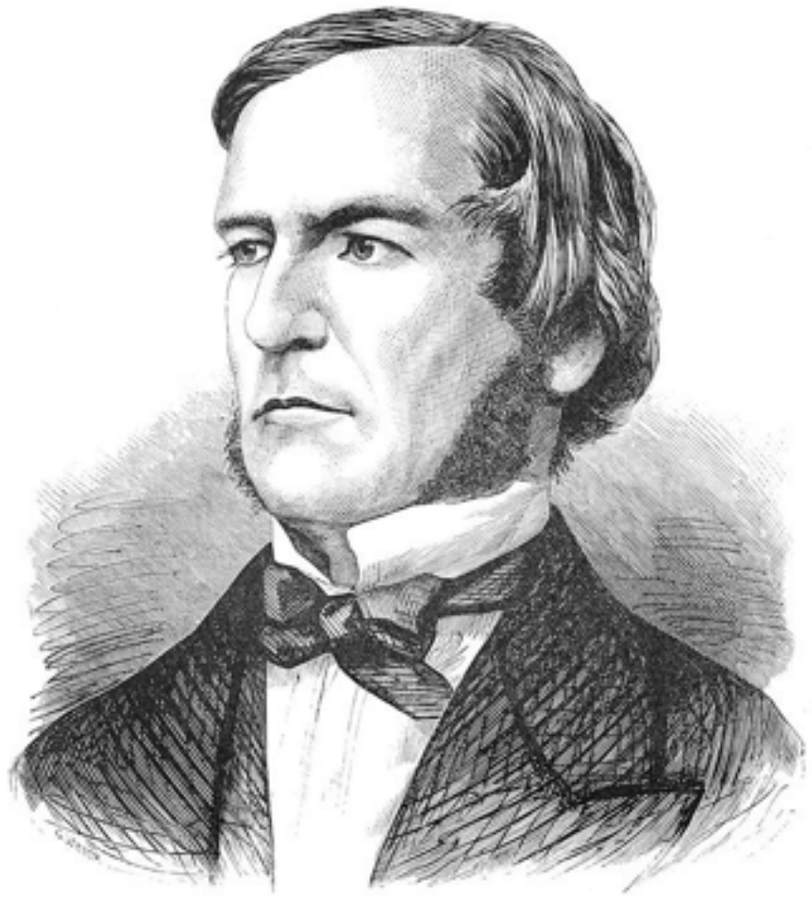
- Boolean data types
- More conditionals
- Find the longest word in a document.


Google
x AND y
x XOR y
x OR y
x
y
NOT y
NOT x

Boolean logic

- George Boole, English mathematician, philosopher, academic.

- Devised a set of “Laws of Thought” which are known as Boolean algebra today, and are the foundations for the information age.



Boolean Values

- Two values — either `True` or `False`
  - (Python needs the capitalisation)

```
>>> x = True
>>> x
True
>>> y = False
>>> y
False
>>> x == y
False
```

Boolean Arithmetic

- Result is always either **`True`** or **`False`**

Note the double equals!

Numerical Arithmetic		
Operator	Example	Result
Sum	1 + 2	3
Difference	1 - 2	-1

Boolean Arithmetic		
Operator	Example	Result
Equals to	`1 == 2`	**`False`**
Not equals to	`1 != 2`	**`True`**
Less than	`1 < 2`	**`True`**
Less than or equal to	`1 <= 2`	**`True`**
Greater than	`1 > 2`	**`False`**
Greater than or equals to	`1 >= 2`	**`False`**

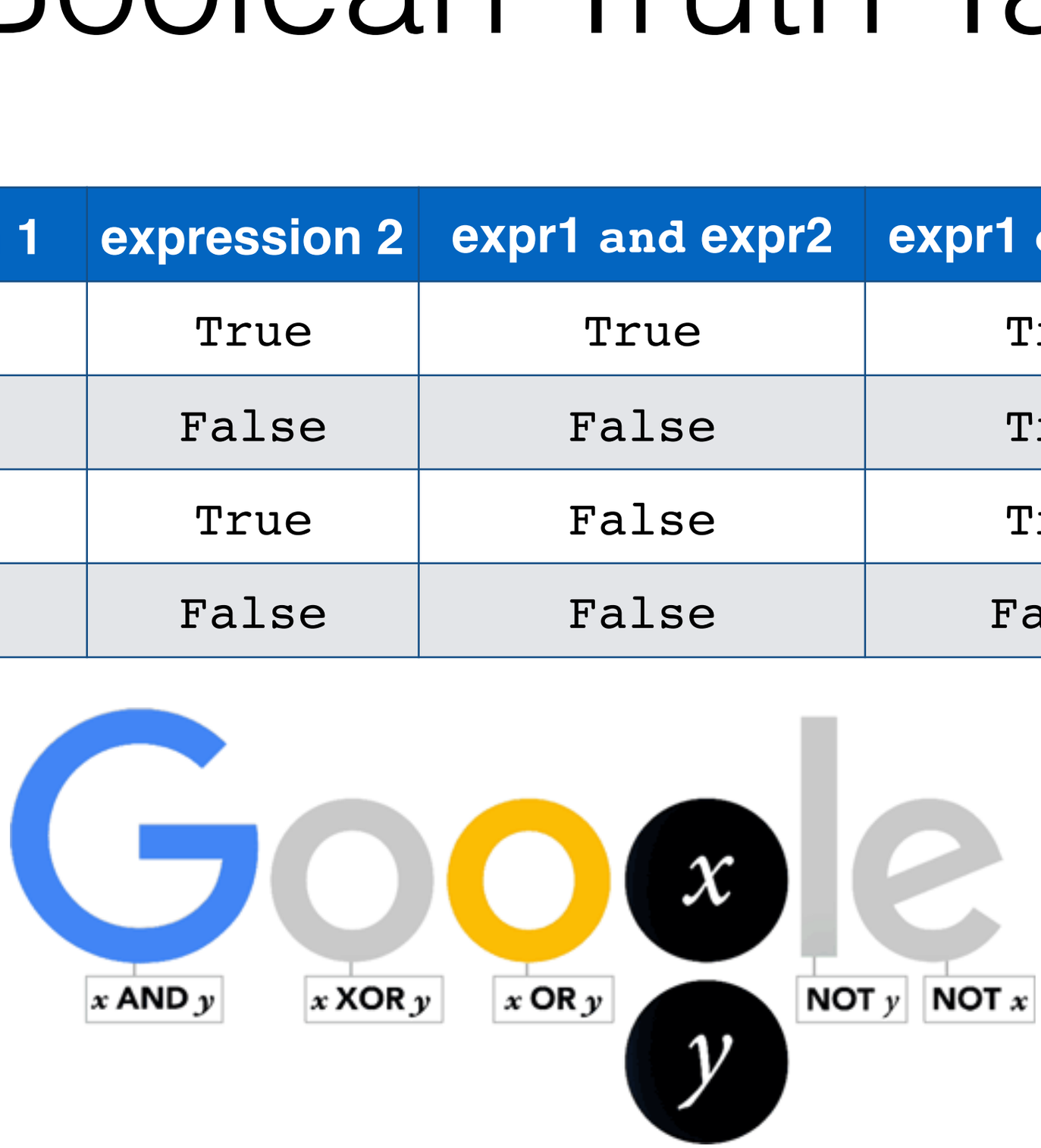
More Boolean Arithmetic

- Three more operators: `and`, `or`, `not`

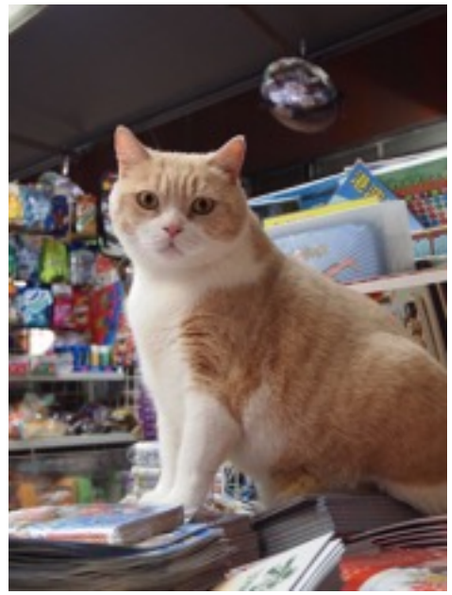
Boolean Arithmetic			
Operator	Example		Result
`and`	`(1 < 2) and (“dog” == “cat”)`	`True and False`	`False`
`or`	`(1 < 2) or (“dog” == “cat”)`	`True or False`	`True`
`not`	`not (“dog” == “cat”)`	`not False`	`True`

Boolean Truth Table

expression 1	expression 2	expr1 `and` expr2	expr1 `or` expr2	`not` expr1
`True`	`True`	`True`	`True`	`False`
`True`	`False`	`False`	`True`	`False`
`False`	`True`	`False`	`True`	`True`
`False`	`False`	`False`	`False`	`True`



Some examples



```
if color == “yellow” and animal == “cat” and location == “TST”:
    name = “Brother Cream”
```

```
if timeLeft < 3 and ourGoals < theirGoals:
    resultOfGame = “lose”
```

```
if (player1Score >= 11 or player2Score >= 11) and
    abs(player1Score - player2Score) >= 2:
    gameOver = True
```

Complex Boolean Expressions

- The last Boolean expression had multiple **`and`** and **`or`** operators in a single expression.

```
if (player1Score >= 11 or player2Score >= 11) and
    abs(player1Score - player2Score) >= 2:
    gameOver = True
```

- There is a precedence order for Boolean operators, just like × and ÷ get evaluated before + and - in arithmetic.

- However, it is best not to try to memorize those rules and just use parentheses (brackets) to do the ordering

Multi-way Conditionals

- So far, our conditional statements have been one-way-only
    - "Do something if the result of a conditional test is **`True`**"
- Python also allows us to do something else if the test is **`False`**

2-way Conditionals

- If something is **`True`**, do A, otherwise, do B

```
if x > y:
   print(x, “is greater than”, y)
else:
   print(x, “is either equals to or smaller than”, y)
```

```
if COMP1D04isFull == False:
   print(“Remember to register for COMP1D04”)
else:
   print(“Find another CAR D subject.”)
```

Multi-way Conditionals

- if something's **`True`**, do A, otherwise, check if something else is True, if that's **`True`**, do B, otherwise, do C

```
if x > y:
   print(x, “is greater than”, y)
elif x == y:
   print(x, “is equals to”, y)
else:
   print(x, “can only be smaller than”, y)
```

Multi-way Conditionals

- Multi-way conditionals evaluate *from the top down*

```
if score > 90:
   grade = “A”
elif score > 80:
   grade = “B”
elif score > 70:
   grade = “C”
elif score > 60:
   grade = “D”
else:
   grade = “F”
```

VS

```
if score > 60:
   grade = “D”
elif score > 70:
   grade = “C”
elif score > 80:
   grade = “B”
elif score > 90:
   grade = “A”
else:
   grade = “F”
```

Do Activity 2-3, Task 1

Linear Search

- Problem: we want to know the longest word in “Tiger Tiger”
- Let’s work on a simpler version of the problem.
- Suppose we have `list1 = [5, 10, -1]`
- We want to find the largest number
- How would we do it?

One way

- One way: use a multi-way `if` statement

```
list1 = [5, 10, -1]

if list1[0] > list1[1] and list1[0] > list1[2]:
   print(list1[0], “is largest”)
elif list1[1] > list1[0] and list1[1] > list1[2]
   print(list1[1], “is largest”)
else:
   print(list1[2], “is largest”)
```

- Fine for now, but what happens if we have a list with 4 elements?
  - Need one more `elif`, and each conditional test needs another comparison
- Not a very ***scalable*** solution!

A better solution

- Imagine holding a shuffled deck of cards.
- You want to find the largest one.
- What would you do?

- Assume first card is the largest.
- Turn over the next card.
- If it's larger than the current largest card, replace the current largest card with this card.
- Repeat last two steps until we have gone through the entire deck

Coding the solution

```
list1 = [5, 10, -1]

# Assume first element is the largest
largestNumber = list1[0]

# Go through all elements in the list
# we will see the first element twice, but that's ok
for item in list1:
    if item > largestNumber:
        largestNumber = item

# Done with the loop. Print out the largest number
print("The largest number is", largestNumber)
```

Initialize-Update-Repeat

- This is a very commonly-used ***programming pattern*** in programming.
- We start by ***initializing*** a variable to some ***initial*** (or starting) value.
- Then we repeat a block of programming statements (usually by looping through a list)
- On each iteration, we ***update*** the variable if needed.
- The value of the variable when we are done looping is the “answer”.

Examples

- If we can find the largest…
    - We can find the smallest
    - Or the longest
    - Or the shortest
    - Or whether an element is in a list (i.e. if a word is in a list of strings.)

Whether an element is in a list

- We start by ***initializing*** a variable to some ***initial*** (or starting) value.
  - `found = False`
- Then we repeat a block of programming statements (usually by looping through a list)
  - `for word in wordlist:`
- On each iteration, we ***update*** the variable if needed.
  - `if word ==` *<word we want>*:
    `found = True`
- The value of the variable when we are done looping is the “answer”.