

Week 2 - Summary

Conditional Statement: if (1/3)

'if' checks if the condition is true and then runs the code after the ':' symbol

code

```
if.py
1  answer = input("Do you want to hear a joke? ")
2
3  ~ if answer == "Yes":
4      print("I'm against picketing, but I don't know how to show it.")
5      # Mitch Hedberg (RIP)
6
```

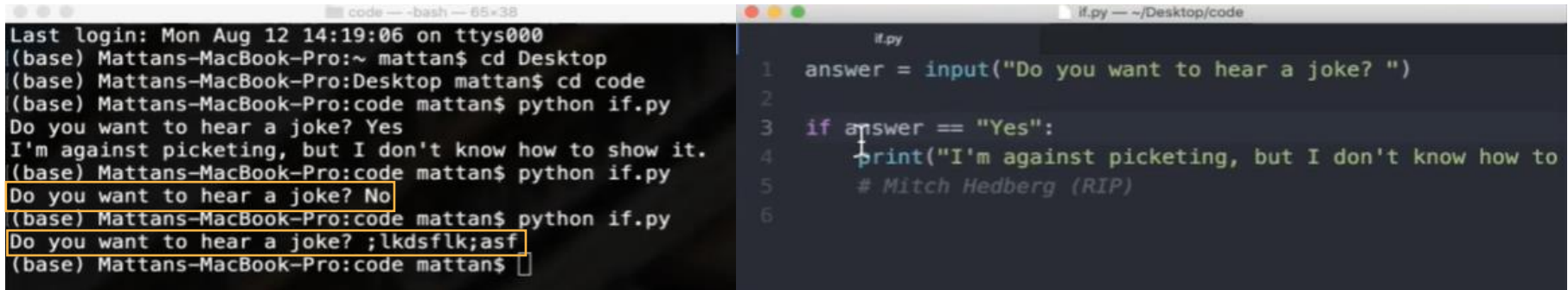
```
code — -bash — 65x38
Last login: Mon Aug 12 14:19:06 on ttys000
(base) Mattans-MacBook-Pro:~ mattan$ cd Desktop
(base) Mattans-MacBook-Pro:Desktop mattan$ cd code
(base) Mattans-MacBook-Pro:code mattan$ python if.py
Do you want to hear a joke? Yes
I'm against picketing, but I don't know how to show it.
```

result(s)

'if' condition validates based on case sensitivity too

Conditional Statement: if (2/3)

If the condition is not met, Python provides no result



```
code — -bash — 65x38
Last login: Mon Aug 12 14:19:06 on ttys000
(base) Mattans-MacBook-Pro:~ mattan$ cd Desktop
(base) Mattans-MacBook-Pro:Desktop mattan$ cd code
(base) Mattans-MacBook-Pro:code mattan$ python if.py
Do you want to hear a joke? Yes
I'm against picketing, but I don't know how to show it.
(base) Mattans-MacBook-Pro:code mattan$ python if.py
Do you want to hear a joke? No
(base) Mattans-MacBook-Pro:code mattan$ python if.py
Do you want to hear a joke? ;lkdsflk;asf
(base) Mattans-MacBook-Pro:code mattan$
```

```
if.py
1  answer = input("Do you want to hear a joke? ")
2
3  if answer == "Yes":
4      print("I'm against picketing, but I don't know how to
5          # Mitch Hedberg (RIP)
6
```

':' symbol allows you to tab the next line to enter the code

'==' symbol checks to see if the two things on the left and right are equal

Conditional Statement: if (3/3)

'=' symbol creates a variable

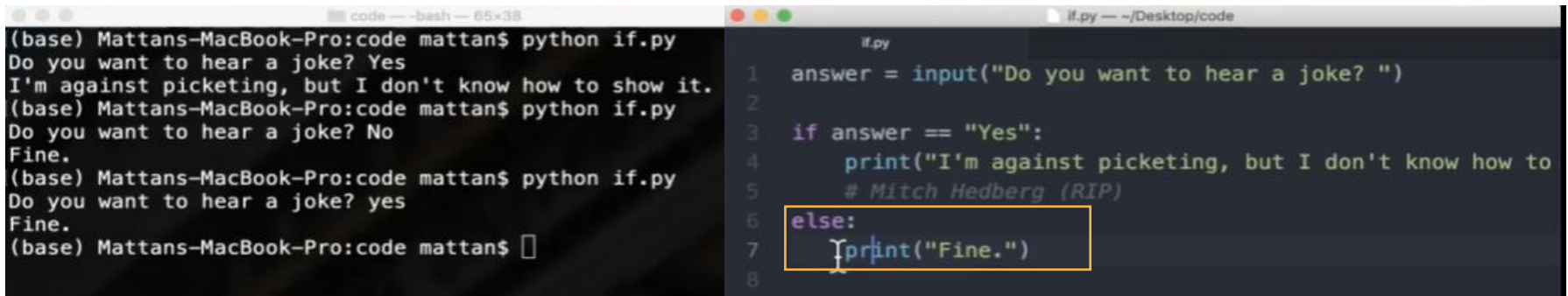
'!=' symbol checks to see if the answer is not equal to the variable

```
Python 3.7.3 (default, Mar 27 2019, 16:54:48)
[Clang 4.0.1 (tags/RELEASE_401/final)] :: Anaconda, Inc.
Type "help", "copyright", "credits" or "license" for more
>>> answer == "Yes"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'answer' is not defined
>>> answer = "Yes"
>>> answer == "Yes"
True
>>> answer = "No"
>>> answer == "Yes"
False
>>> answer != "Blue"
True
>>> answer != "No"
False
```

Variables can be rewritten in Python

Conditional Statement: else and elif (1/2)

'else' condition is triggered when the 'if' condition is not met



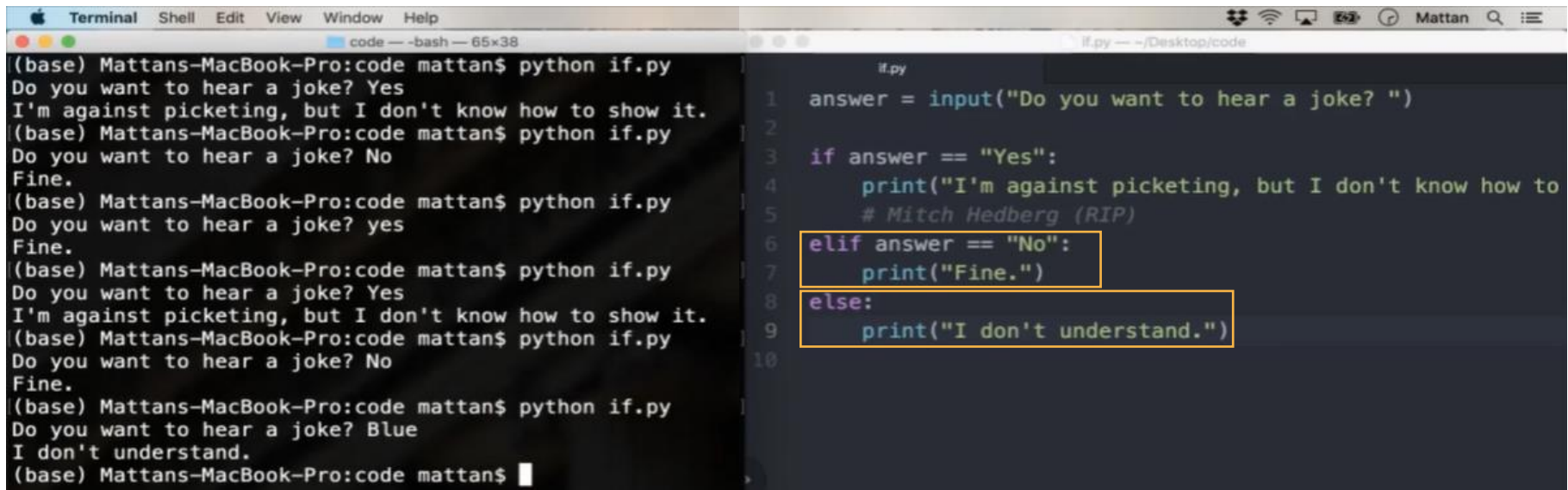
The image shows two side-by-side windows. The left window is a terminal titled 'code — -bash — 65x38' showing the execution of a Python script named 'if.py'. The user runs 'python if.py' three times, providing different inputs: 'Yes', 'No', and 'yes'. The script outputs 'I'm against picketing, but I don't know how to show it.' for 'Yes' and 'Fine.' for 'No' and 'yes'. The right window is a code editor titled 'if.py — ~/Desktop/code' showing the source code of the script. The code is as follows:

```
if.py
1  answer = input("Do you want to hear a joke? ")
2
3  if answer == "Yes":
4      print("I'm against picketing, but I don't know how to
5          # Mitch Hedberg (RIP)
6  else:
7      print("Fine.")
8
```

The 'else:' line in the code editor is highlighted with a yellow box.

Conditional Statement: else and elif (2/2)

'elif' allows you to set a second 'if' condition



```
Terminal Shell Edit View Window Help
code — bash — 65x38
(base) Mattans-MacBook-Pro:code mattan$ python if.py
Do you want to hear a joke? Yes
I'm against picketing, but I don't know how to show it.
(base) Mattans-MacBook-Pro:code mattan$ python if.py
Do you want to hear a joke? No
Fine.
(base) Mattans-MacBook-Pro:code mattan$ python if.py
Do you want to hear a joke? yes
Fine.
(base) Mattans-MacBook-Pro:code mattan$ python if.py
Do you want to hear a joke? Yes
I'm against picketing, but I don't know how to show it.
(base) Mattans-MacBook-Pro:code mattan$ python if.py
Do you want to hear a joke? No
Fine.
(base) Mattans-MacBook-Pro:code mattan$ python if.py
Do you want to hear a joke? Blue
I don't understand.
(base) Mattans-MacBook-Pro:code mattan$

if.py
1  answer = input("Do you want to hear a joke? ")
2
3  if answer == "Yes":
4      print("I'm against picketing, but I don't know how to
5          # Mitch Hedberg (RIP)
6  elif answer == "No":
7      print("Fine.")
8  else:
9      print("I don't understand.")
10
```

Conditional Statements: if, else and elif

'if' is independent of 'elif' and 'else'

All the three conditions have to be in the order of - 'if', 'elif' and 'else'

There can be several 'elif's but only one 'else'



Operators in Python

To produce True or False statement

Three additional terms are *not*, *and*, and *or*.

Operator	Description
=	equal to
!=	not equal to
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to

Truth Table: not

Code (not)	Result
not True	False
not False	True

True and False start with capital letters in Python to make them work properly.

Truth Table: and

Code (and)	Result
True and True	True
True and False	False
False and True	False
False and False	False

Truth Table: or

Code (or)	Result
True and True	True
True and False	True
False and True	True
False and False	False

Logical Function: or

Any string value in Python is, by default, true

```
if.py
1  answer = input("Do you want to hear a joke? ")
2
3  if answer == "Yes" or answer == "yes":
4      print("I'm against picketing, but I don't know how to
5          # Mitch Hedberg (RIP)
6  elif answer == "No" or answer == "no":
7      print("Fine.")
8  else:
9      print("I don't understand.")
10
```

Lists

Lists are a way of grouping similar things together. They start and end with square brackets and have 'elements' in between.

```
lists.py -- ~/Desktop/code
1  # In Python, lists are ways of grouping together
2  # similar things (usually)
3  the_count = [1, 2, 3, 4, 5]
4  stocks = ["FB", "AAPL", "NFLX", "GOOG"]
5  random_things = [55, 1/2, "Puppies", stocks]
6
7  people = []
8
9  people.append("Mattan")
10 people.append("Daniel")
11 people.append("Sam")
12 people.remove("Daniel")
13
14 print(people)
```

Creating a list

Use '<listname>.append' to add entries to the list and '<listname>.remove' to remove them

More Ways to Create Lists

```
16 print("New York, San Francisco, London".split(", "))
17
```

```
['New York', 'San Francisco', 'London']
(base) Mattans-MacBook-Pro:code mattan$
```

Splitting a list of items based on specific conditions

```
16 print("New York, San Francisco, London".split(", "))
17 print(", ".join(["Milk", "Eggs", "Cheese"]))
18
```

```
Milk, Eggs, Cheese
(base) Mattans-MacBook-Pro:code mattan$
```

Joining items to make a list

Accessing Elements of a List

You can access elements of a list using square brackets []

```
16 print("New York, San Francisco, London".split(", "))  
17
```

```
first_city = cities[0]  
second_city = cities[1]  
last_city = cities[-1]
```

```
first_two_cities = cities[0:2]  
print(first_two_cities)
```

```
['New York', 'San Francisco']  
(base) Mattans-MacBook-Pro:code mattan$
```

The first item of the list is numbered as zero and to access the last item in the list use '-1'

Slice notation refers to accessing parts of a list. Here '0:2' tells Python to show cities from the first until the third one but excluding the last.

Loops in Lists (1/2)

- A loop is a faster and more concise way of repeating an action
- Use 'for' to loop over lists

```
(base) Mattans-MacBook-Pro:code mattan$ python loops.py
1
2
3
(base) Mattans-MacBook-Pro:code mattan$ python loops.py
1
2
3
1
2
3
(base) Mattans-MacBook-Pro:code mattan$ python loops.py
1
2
3
1
2
3
```

```
loops.py
1  # Use for to loop over a list
2  numbers = [1, 2, 3]
3  for number in numbers:
4      print(number)
5
6  numbers = [1, 2, 3]
7  number = numbers[0]
8  print(number)
9  number = numbers[1]
10 print(number)
11 number = numbers[2]
12 print(number)
13
```


Loops in Lists (2/2)

While looping, you can also change the casing of the items in the list

```
FB  
AAPL  
NFLX  
GOOG  
(base) Mattans-MacBook-Pro:code mattan$
```

```
14 stocks = ["fb", "aapl", "nflx", "goog"]  
15 for stock in stocks:  
16     print(stock.upper())  
17
```

Append Within a For Loop

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361, 400, 441, 484, 529, 576, 625, 676, 729, 784, 841, 900, 961, 1024, 1089, 1156, 1225, 1296, 1369, 1444, 1521, 1600, 1681, 1764, 1849, 1936, 2025, 2116, 2209, 2304, 2401, 2500, 2601, 2704, 2809, 2916, 3025, 3136, 3249, 3364, 3481, 3600, 3721, 3844, 3969, 4096, 4225, 4356, 4489, 4624, 4761, 4900, 5041, 5184, 5329, 5476, 5625, 5776, 5929, 6084, 6241, 6400, 6561, 6724, 6889, 7056, 7225, 7396, 7569, 7744, 7921, 8100, 8281, 8464, 8649, 8836, 9025, 9216, 9409, 9604, 9801, 10000]
```

```
22 # You can do more than just print in a for loop
23 squares = []
24 for number in range(1, 101):
25     squares.append(number**2)
26 print(squares)
27
```

Creating a List From Another List

List comprehension is when we create a list using an existing list

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361, 400, 441, 484, 529, 576, 625, 676, 729, 784, 841, 900, 961, 1024, 1089, 1156, 1225, 1296, 1369, 1444, 1521, 1600, 1681, 1764, 1849, 1936, 2025, 2116, 2209, 2304, 2401, 2500, 2601, 2704, 2809, 2916, 3025, 3136, 3249, 3364, 3481, 3600, 3721, 3844, 3969, 4096, 4225, 4356, 4489, 4624, 4761, 4900, 5041, 5184, 5329, 5476, 5625, 5776, 5929, 6084, 6241, 6400, 6561, 6724, 6889, 7056, 7225, 7396, 7569, 7744, 7921, 8100, 8281, 8464, 8649, 8836, 9025, 9216, 9409, 9604, 9801, 10000]  
['FB', 'AAPL', 'NFLX', 'GOOG']
```

```
23 squares = []  
24 for number in range(1, 101):  
25     squares.append(number**2)  
26 print(squares)  
27  
28 # List comprehension  
29 print([stock.upper() for stock in stocks])  
30
```