



DATUM

Yonsei Data Science Academia

VARIABLES, EXPRESSIONS AND STATEMENTS

CONSTANTS

Fixed values such as numbers, letters, and strings are called “constants” because their value does not change. We use this to do calculations etc
(string constants use single quotes (‘) or double quotes (“))

CODING EXAMPLE

```
>>> print(123)  
123  
>>> print(98.6)  
98.6  
>>> print("Hello World")  
Hello World
```

CONSTANTS

ENGLISH EXAMPLE

If your work time is more than
40 hours, we will do X

RESERVED WORDS

YOU CANNOT USE RESERVED WORDS AS VARIABLE NAMES / IDENTIFIERS

False, None, True, and, as, assert, break, class, if, def, del, elif, else, except, return, for, from, global, try, import, in, is, lambda, while, not, or, pass, raise, finally, continue, nonlocal, with, yield

These words have special meaning embedded to them. **For example**, if you use the word “if”, python thinks to make it a conditional rather than accepting the word “it” as name.

VARIABLES

A variable is a named place where a programmer can store data and later retrieve the data using the variable
"name"

CODING EXAMPLE

```
>>> x=12.2
>>> y=14
>>> print(x)
12.2
>>> print(y)
14
```

Explaining in plain English

EXPLANATION

Assign constant 12.2 to "x"

Assign constant 14 to "y"

Show me what "x" is

Show me what "y" is

VARIABLES

Variables can only hold one value.

So if you assign a new value, it forgets the old one.

The fact that you can change is the reason why it's called a
variable

CODING EXAMPLE

```
>>> x=12.2
>>> y=14
>>> x=55
>>> print(x)
55
>>> print(y)
14
```

Explaining in plain English

EXPLANATION

Assign constant 12.2 to "x"

Assign constant 14 to "y"

Assign constant 55 to "x"

Show me what "x" is

Show me what "y" is

VARIABLES

Variables can only hold one value.

So if you assign a new value, it forgets the old one.

The fact that you can change is the reason why it's called a
variable

CODING EXAMPLE

```
>>> x="what is the answer to life?"  
>>> y=50  
>>> z=8  
>>> print(x, y-z)  
what is the answer to life? 42
```

EXPLANATION

Assign constant "what is the answer to life" to "x"

Assign constant 50 to "y"

Assign constant 8 to "x"

Show me what "x" and "y-z" is

PYTHON NAMING RULES

Must consist of letters, numbers, and underscores

Case sensitive

Must start with a letter or an underscore (very rare occasion)

Good

Spam
Eggs
Spam23
_speed

Bad

23spam
#deep
Ver.12

Different

spam
Spam
SPAM

SNOBBISH PYTHON NAMING RULES

Usually choose our variable names so we can remember what the variable is later on.

BAD

```
>>> asdfjkl=35.0
>>> asdfjhl=12.50
>>> asdfjlk=asdfjkl * asdfjhl
>>> print(asdfjlk)
437.5
```

EH

```
>>> a= 35.0
>>> b= 12.50
>>> c= a * b
>>> print(c)
437.5
```

BETTER

```
>>> hours=35.0
>>> rate=12.50
>>> pay=hours*rate
>>> print(pay)
437.5
```

ASSIGNMENT STATEMENTS

We assign a value to a variable using the assignment statement (=)

An assignment statement consists of an expression on the right-hand side and a variable to store the result

```
>>> x=3.9*15*(1-x)
```

VARIABLE ON BOTH SIDES

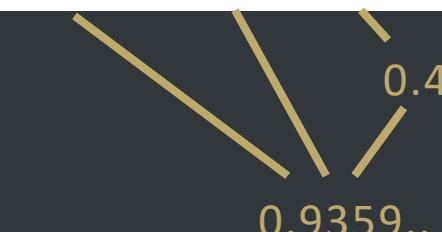
ASSIGNMENT STATEMENTS

Right side is an expression

Once the expression is evaluated, the result is placed in (assigned to) x

```
>>> x=0.6
>>> x=3.9*x*(1-x)
>>> print(x)
0.9359999999999999
```

```
>>> x=3.9*15*(1-x)
```



0.4
0.9359..

“=” SIGN DOES NOT MEAN
“EQUAL TO” IN PYTHON!

It's more like a ->

EXPRESSIONS

Numeric Expression help us express classic math operators.

Order of Precedence Rules

```
>>> x=2
>>> y=5
>>> x+y
7
>>> x-y
-3
>>> x*y
10
>>> x/y
0.4
>>> x**y
32
>>> x%y
2
```

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Power
%	Remainder

PEMDAS

A X U I D U
R P L V D B
E O T I I T
N E N I S T R
T E P I I A
N T N L O O C
H T I N N T
E I C I
S A A O
T T T n
E I I O
O O N
N N N

TYPE

PYTHON KNOWS THE DIFFERENCE BETWEEN NUMBER AND A STRING

For example, “+” means “addition” if something is a number and “concatenate” if something is a string.

```
>>> x=1+4
>>> print(x)
5
>>> y="python"+"sucks"
>>> print(y)
pythonsucks
```

ADDITIONS

CONCATENATE (MEANS PUT TOGETHER)

TYPE

PYTHON KNOWS WHAT “TYPE” EVERY THING IS.

If you don't know what type something is, asking python

```
>>> y="python"+"sucks"  
>>> y=y+1  
Traceback (most recent call last):  
  File "<pyshell#83>", line 1, in <module>  
    y=y+1  
TypeError: can only concatenate str (not 'int') to str
```

put together

string

integer

TYPE

PYTHON KNOWS WHAT “TYPE” EVERY THING IS.

If you don't know what type something is, asking python

```
>>> type(y)
<class 'str'>
>>> type(1)
<class 'int'>
```

TYPE OF NUMBERS

NUMBERS HAVE TWO
MAIN TYPES

Integers as whole numbers :
-14, -2, 0, 1, 100

Floating point numbers :-
2.5, 0.0, 14.0, 111.1

There are other number
types

```
>>> type(1)
<class 'int'>
>>> type(1.0)
<class 'float'>
>>> type(-2.0)
<class 'float'>
>>> type(-2)
<class 'int'>
```

TYPE CONVERSIONS

WHEN YOU PUT INTEGER AND FLOATING IN AN EXPRESSION, IT TURNS IN TO A FLOAT

Control this by using built-in functions `int()` and `float()`

TYPE CONVERSIONS

Turned 99 into a float

```
>>> print(float(99) + 100)
199.0
>>> i = 42
>>> type(i)
<class 'int'>
>>> f = float(i)
>>> print(f)
42.0
>>> type(f)
<class 'float'>
```

INTEGER DIVISION

Integer division produces a floating point result

```
>>> print(10/2)
5.0
>>> print(9/2)
4.5
```

STRING CONVERSIONS

USE INT() AND FLOAT() TO CONVERT BETWEEN STRING AND INTEGERS

Python will get grumpy and produce an error if string does not contain numeric characters

```
>>> yolo="123"
>>> type(yolo)
<class 'str'>
>>> yolt=int(yolo)
>>> type(yolt)
<class 'int'>
```

EXPLANATION

Assign string “123” to “yolo”

Ask python what type yolo is

Python replies that it is a string

Type conversion “yolo” to an integer and assign that to “yolt”

Ask python what type yolt is

It is an integer

USER INPUT

Instruct python to pause and read data using `input()` function- this returns as string

```
>>> nam = input("Who are you? ")  
Who are you? Chuck  
>>> print("Welcome", nam)  
Welcome Chuck
```

ANYTHING AFTER A # IS IGNORED IN PYTHON

Used to describe code
Turn off line of code

```
1 #assign value 12 to x
2 x = 12
3
4 #assign value 10 to y
5 y= 10
6
7 #example of addition
8 x + y |
```

```
1 #get the name of the file and open it
2 name = input("Enter file:")
3 handle = open(name, 'r')
4
5 #count word frequency
6 counts = dict()
7 for line in handle:
8     words = line.split()
9     for word in words:
10         counts[word] = counts.get(word, 0) + 1
11
```

```
11
12     #find the most common word
13     bigcount = None
14     Bigword = None
15     for word, count in counts.items():
16         if bigcount is None or count > bigcount:
17             bigword = word
18             bigcount = count
19
20     #All done
21     print(bigword, bigcount)
```

SUMMARY



TYPE

Float, integers, strings

Floats are decimals (1.0, 1.1)

Integers are whole numbers

Strings are letters



OPERATORS

+, -, *, &, %, ** are used to demonstrate classical mathematical problems



CONVERSION BETWEEN TYPES

You can use type() to find out what type your constant is and use float(), int(), str() to turn them into the type you want



RESERVED WORDS

Some words serve a function in python (for example: type, if, else) and should not be used when naming a variable



OPERATOR PRECEDENCE

Follows PEMDAS like any other mathematical formula



USER INPUT

Input() function means it waits for you to input a value, assigning it to the variable.



VARIABLES

Naming a variable should be intuitive.

Variable can be only assigned to one thing.



ASSIGNMENT VARIABLES

In python, you can have X's on both side and have the X contain different values depending on which side they are at.



COMMENTS

Using #, you can put comments on a line of code without affecting the code itself.

CHALLENGE

IN THE US, THE ELEVATOR NUMBERING SYSTEM STARTS FROM 1. IN EUROPE, IT STARTS FROM 0.

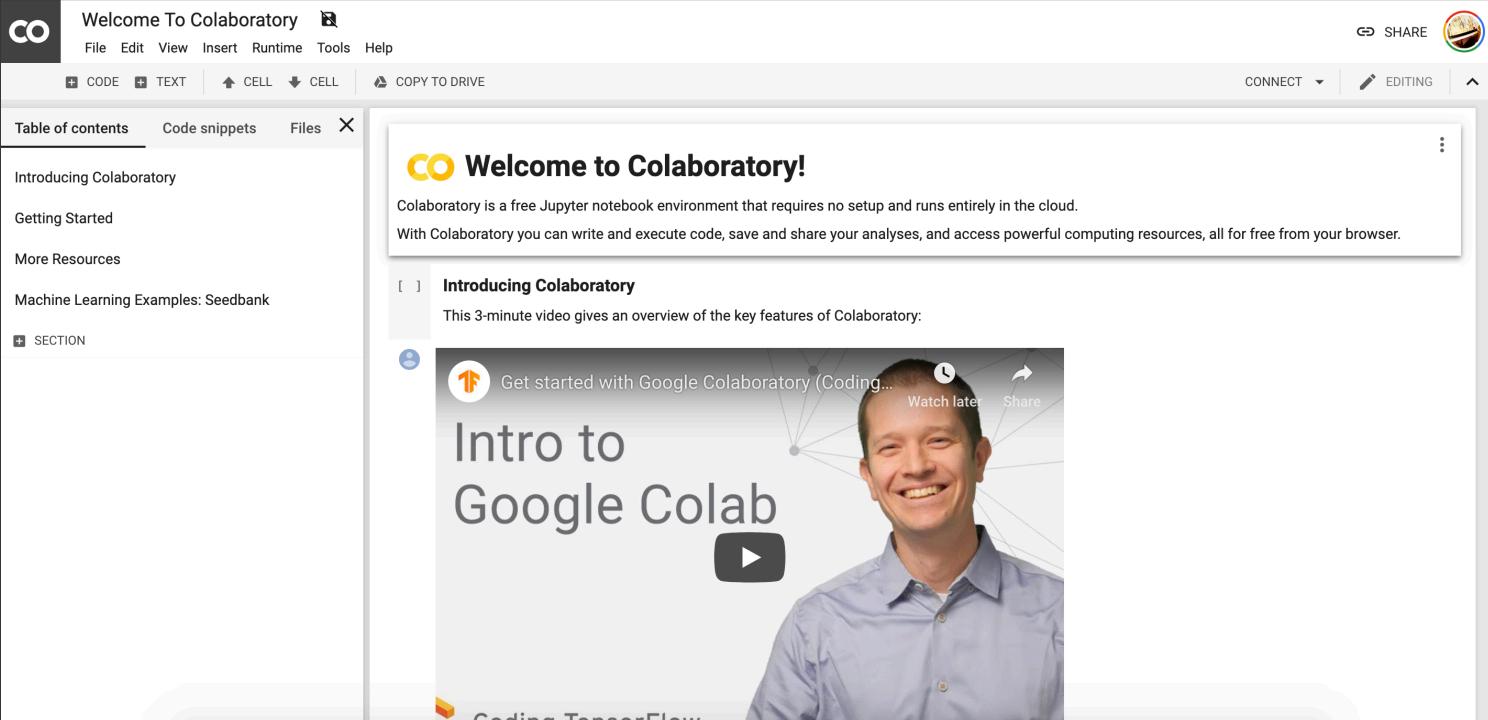
I WANT TO MAKE A LINE OF CODE THAT HELPS ME FIGURE OUT WHICH US FLOOR CORRESPONDS TO THE EUROPE FLOOR.

SO, A LINE OF CODE THAT CONVERTS EUROPE FLOOR INTO US FLOOR.

CHALLENGE

- ✓ It should ask the person to input the floor they would like to convert.
- ✓ Difference between European Floor and US floor is always 1, with US floor being 1 more.
- ✓ In order to perform addition, it needs to be an integer
- ✓ Print out the answer to the person so they can see.

Go to coloab.com



Welcome To Colaboratory

File Edit View Insert Runtime Tools Help

SHARE

CODE TEXT CELL COPY TO DRIVE CONNECT EDITING

Table of contents Code snippets Files

Introducing Colaboratory

Getting Started

More Resources

Machine Learning Examples: Seedbank

SECTION

Welcome to Colaboratory!

Colaboratory is a free Jupyter notebook environment that requires no setup and runs entirely in the cloud.

With Colaboratory you can write and execute code, save and share your analyses, and access powerful computing resources, all for free from your browser.

Introducing Colaboratory

This 3-minute video gives an overview of the key features of Colaboratory:

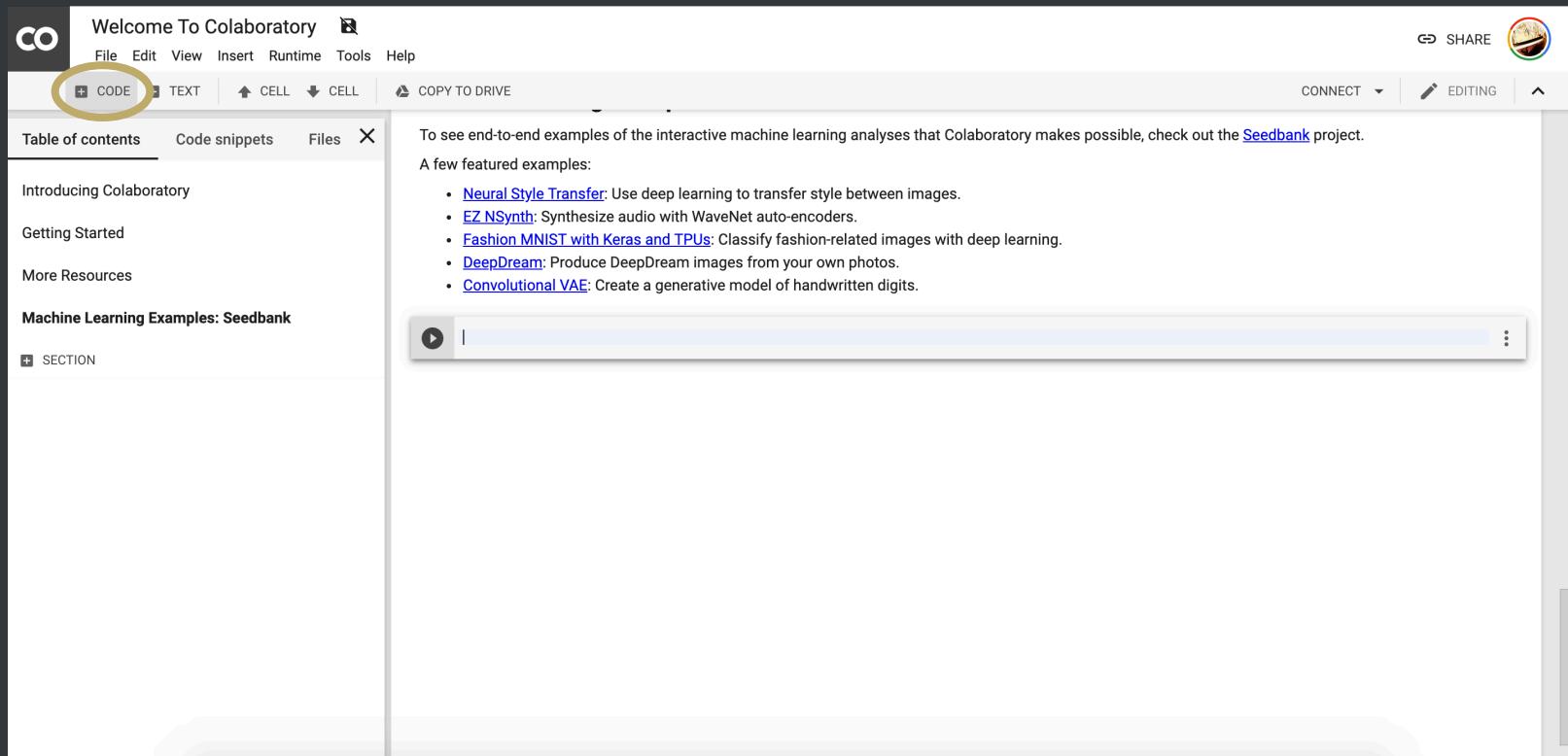
Get started with Google Colaboratory (Coding...)

Watch later Share

Intro to Google Colab

Coding TensorFlow

Go to coloab.com



Welcome To Colaboratory

File Edit View Insert Runtime Tools Help

SHARE 

CODE TEXT  COPY TO DRIVE

CONNECT EDITING

Table of contents Code snippets Files X

Introducing Colaboratory

Getting Started

More Resources

Machine Learning Examples: Seedbank

SECTION

To see end-to-end examples of the interactive machine learning analyses that Colaboratory makes possible, check out the [Seedbank](#) project.

A few featured examples:

- [Neural Style Transfer](#): Use deep learning to transfer style between images.
- [EZ NSynth](#): Synthesize audio with WaveNet auto-encoders.
- [Fashion MNIST with Keras and TPUs](#): Classify fashion-related images with deep learning.
- [DeepDream](#): Produce DeepDream images from your own photos.
- [Convolutional VAE](#): Create a generative model of handwritten digits.

Go to coloab.com

The screenshot shows the Google Colaboratory interface. At the top, there's a navigation bar with the 'Welcome To Colaboratory' logo, 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', 'Help' menus, and 'SHARE' and 'EDITING' buttons. A yellow circle highlights the 'CODE' button in the top bar. On the left, a sidebar has a 'CODE' tab (also highlighted with a yellow circle), 'TEXT', 'CELL', 'COPY TO DRIVE', 'Table of contents', 'Code snippets', 'Files', 'Introducing Colaboratory', 'Getting Started', 'More Resources', 'Machine Learning Examples: Seedbank', and 'SECTION' buttons. The main content area displays a list of featured examples: 'Neural Style Transfer', 'EZ NSynth', 'Fashion MNIST with Keras and TPUs', 'DeepDream', and 'Convolutional VAE'. Below this is a code cell with a play button and a text input field. The code cell contains the line `print("Yes")`, and the output shows the word 'Yes'. A yellow circle highlights the play button in the code cell.

SOLUTION

```
ef = input("Europe Floor?")
usf = int(ef) + 1
print("US Floor", usf)
```

```
Europe Floor?0
US Floor 1
```



DATUM

Yonsei Data Science Academia

CONDITIONAL EXECUTION

CONDITIONAL STEPS

```
x=5
if x<10:
    print("Smaller")
if x>20:
    print("Bigger")

print("Finish")
```

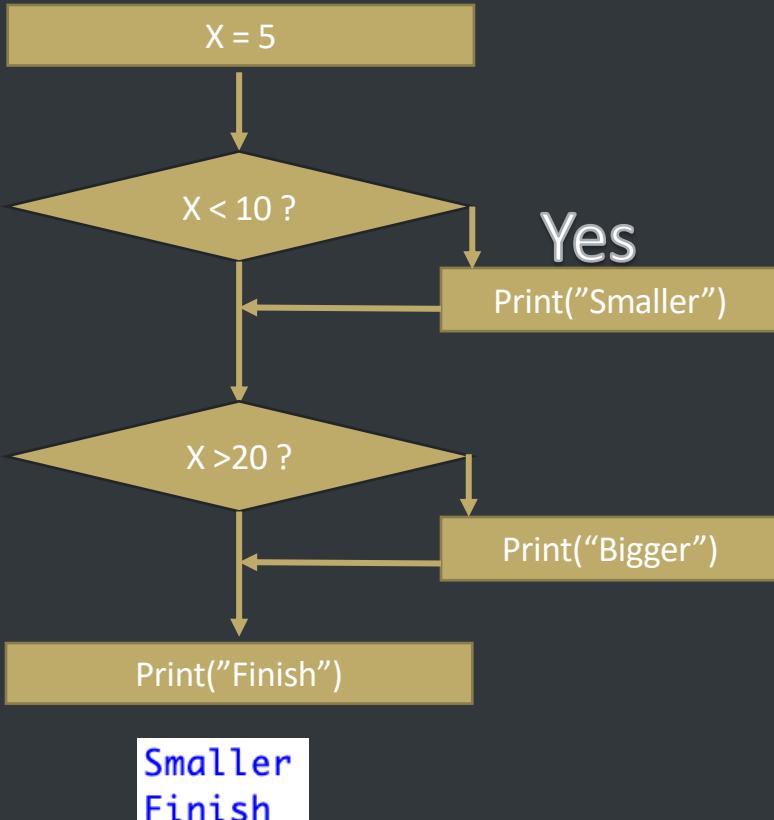
CONDITIONAL STEPS

```
x=5
if x<10:
    print("Smaller")
if x>20:
    print("Bigger")
print("Finish")
```

CONDITIONAL STEPS

```
x=5
if x<10:
    print("Smaller")
if x>20:
    print("Bigger")

print("Finish")
```



COMPARISON OPERATORS

Boolean expression ask a question and produce yes or no which we use to control program flow.

Python	Meaning
<	Less than
<=	Less than or Equal to
==	Equal to
>=	Greater than or Equal to
>	Greater than
!=	Not equal

REMEMBER THAT “=” IS USED FOR ASSIGNMENT, THEREFORE THE EQUAL TO IS “==”

COMPARISON OPERATORS

Examples of one-way decisions

```
x = 5

print("What value is X?")

if x==5 :
    print("It is 5")

if x==6 :
    print("It is 6")

print("Thank you")
```

COMPARISON OPERATORS

Examples of one-way decisions

```
x = 5

print("What value is X?")

if x==5 :
    print("It is 5")

if x==6 :
    print("It is 6")

print("Thank you")
```

COMPARISON OPERATORS

Examples of one-way decisions

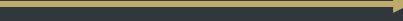
```
x = 5

print("What value is X?")

if x==5 :
    print("It is 5")

if x==6 :
    print("It is 6")

print("Thank you")
```



```
What value is X?
It is 5
Thank you
```

INDENTATION

Increase indent after an if statement or for statement (after:)

Maintain indent to indicate the **scope** of a block (Which lines are
affected by the if/for)

Reduce indent when if/for statement to indicate end of block

Blank lines are **ignored**, they are not indentation

INDENTATION

```
x = 5
if x > 2 :
    print("Bigger than 2")
    print("Still bigger")
print("Done with 2")

for i in range(5) :
    print(i)
    if i > 2:
        print("Bigger than 2")
    print("Done with i", i)
print("All Done! ")
```

INDENTATION

```
x = 5
if x > 2 :
    print("Bigger than 2")
    print("Still bigger")
print("Done with 2")
```

```
for i in range(5) :
    print(i)
    if i > 2:
        print("Bigger than 2")
    print("Done with i", i)
print("All Done! ")
```

INDENTATION

```
x = 5
if x > 2 :
    print("Bigger than 2")
    print("Still bigger")
print("Done with 2")

for i in range(5) :
    print(i)
    if i > 2:
        print("Bigger than 2")
    print("Done with i", i)
print("All Done! ")
```

INDENTATION

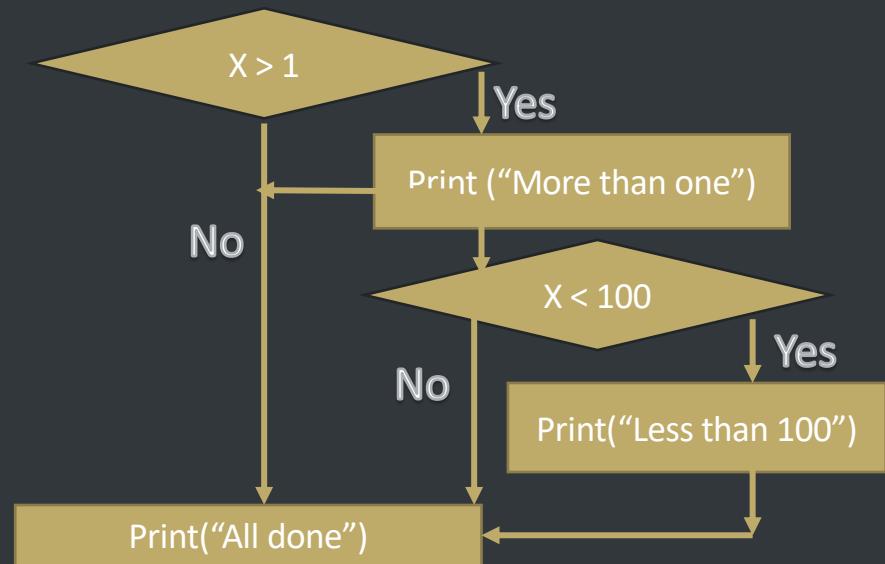
```
x = 5
if x > 2 :
    print("Bigger than 2")
    print("Still bigger")
print("Done with 2")                                Block 1

for i in range(5) :
    print(i)
    if i > 2:
        print("Bigger than 2")
        print("Done with i", i)                      Block 2

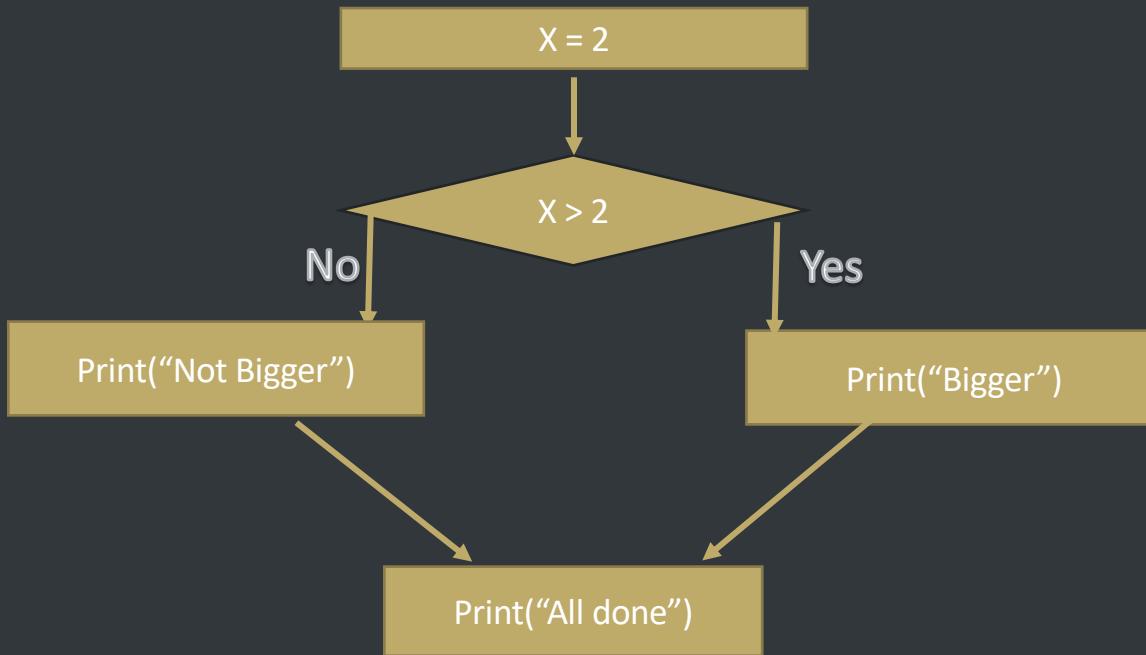
print("All Done! ")                                  Block 3
```

NESTED DECISIONS

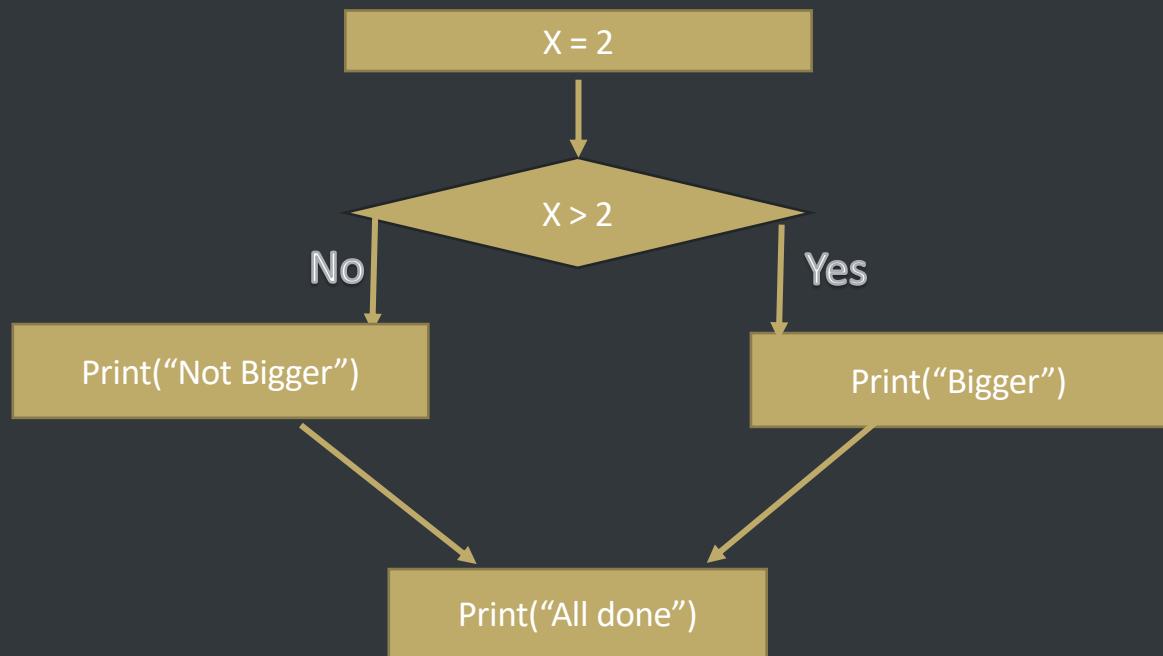
```
x = 42
if x > 1 :
    print("More than one")
    if x < 100 :
        print("Less than 100")
print("All Done")
```



TWO – WAY DECISIONS



IF ELSE

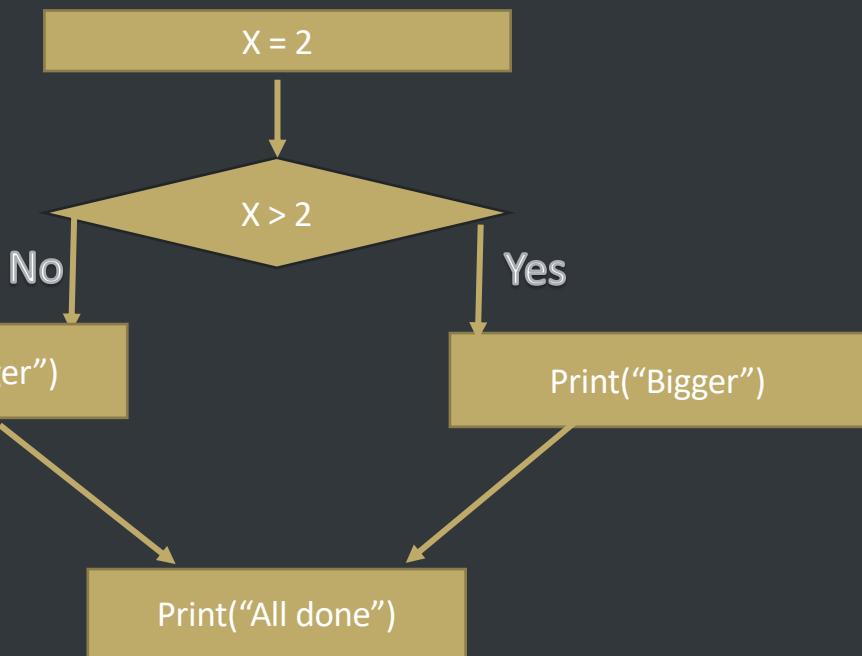


```
x = 4

if x > 2:
    print("Bigger")
else :
    print("Smaller")

print ("All done")
```

IF ELSE



```
x = 4
```

When true

```
if x > 2:  
    print("Bigger")
```

```
else :  
    print("Smaller")
```

When false

```
print ("All done")
```

MULTI-WAY

ELIF

Elif is another reserved word in python

It checks the first, and if true, only runs that function.

```
if x < 2 :  
    print("Small")  
elif x < 10 :  
    print("Medium")  
else:  
    print("Large")  
print("All Done")
```

MULTI-WAY

ELIF

Elif is another reserved word in python

It checks the first, and if true, only runs that function.

```
if x < 2 :  
    print("Small")  
elif x < 10 :  
    print("Medium")  
else:  
    print("Large")  
print("All Done")
```



MULTI-WAY

ELIF

Elif is another reserved word in python

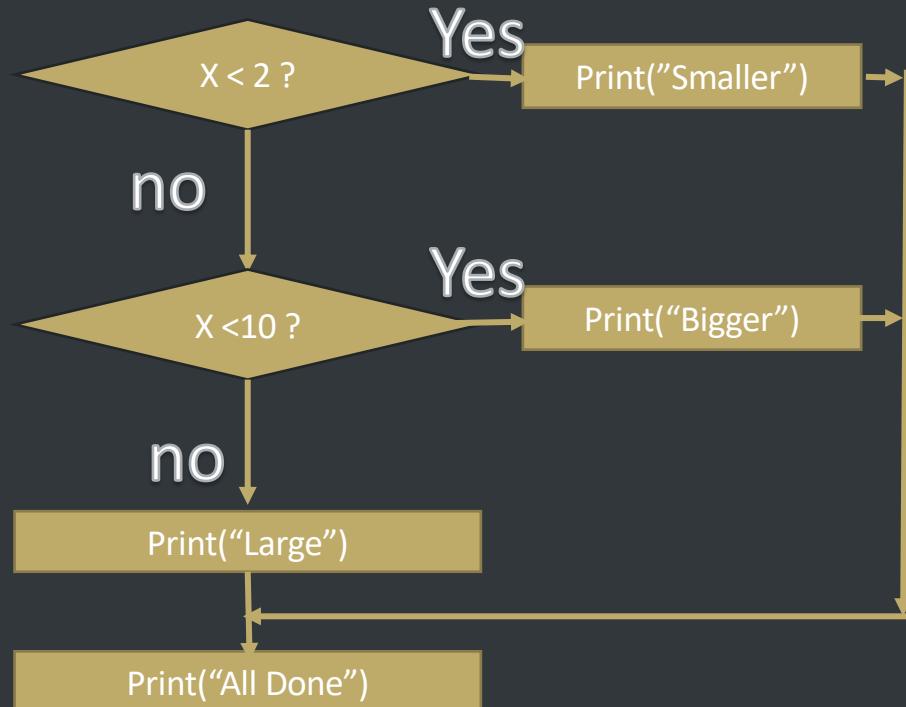
It checks the first, and if true, only runs that function.

```
if x < 2 :  
    print("Small")  
elif x < 10 :  
    print("Medium")  
else:  
    print("Large")  
print("All Done")
```



WHEN X = 5

```
if x < 2 :  
    print("Small")  
elif x < 10 :  
    print("Medium")  
else:  
    print("Large")  
print("All Done")
```

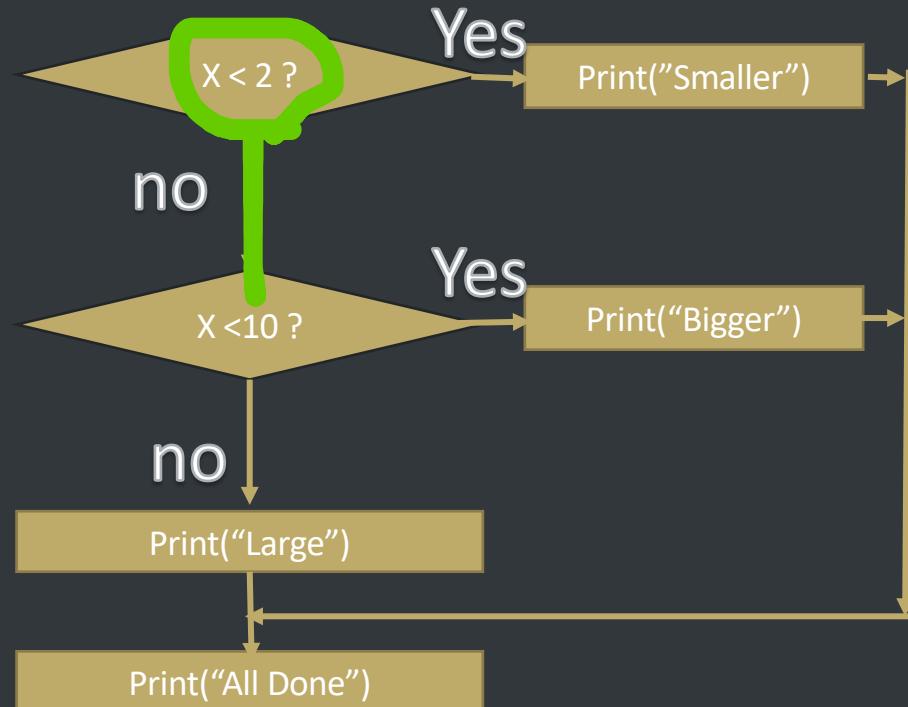


```

if x < 2 :
    print("Small")
elif x < 10 :
    print("Medium")
else:
    print("Large")
print("All Done")

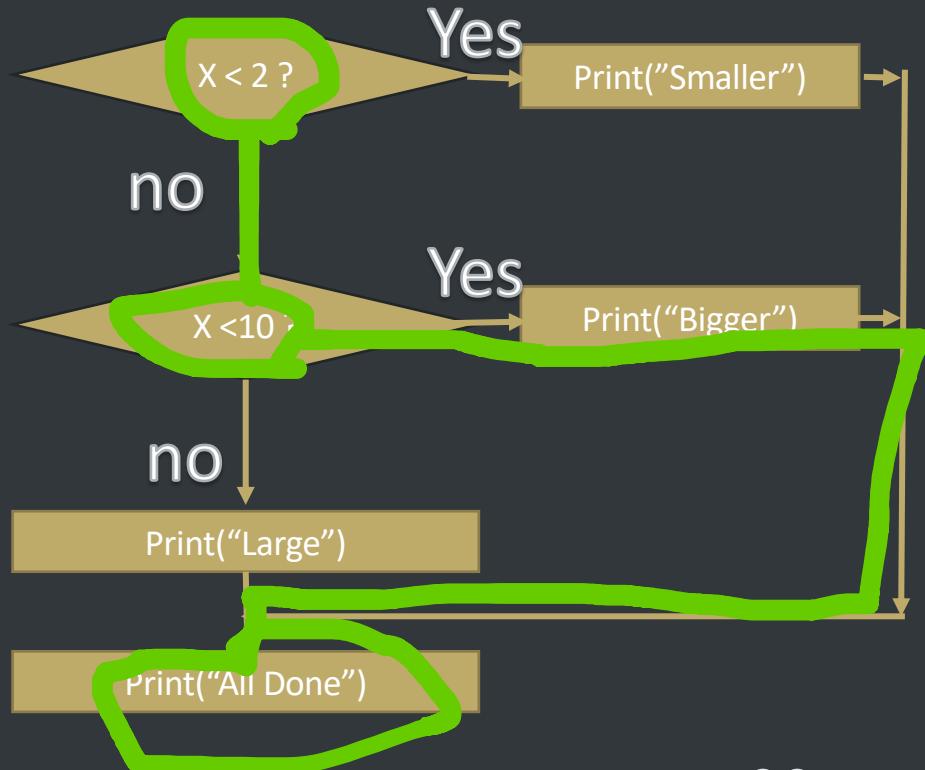
```

WHEN X = 5



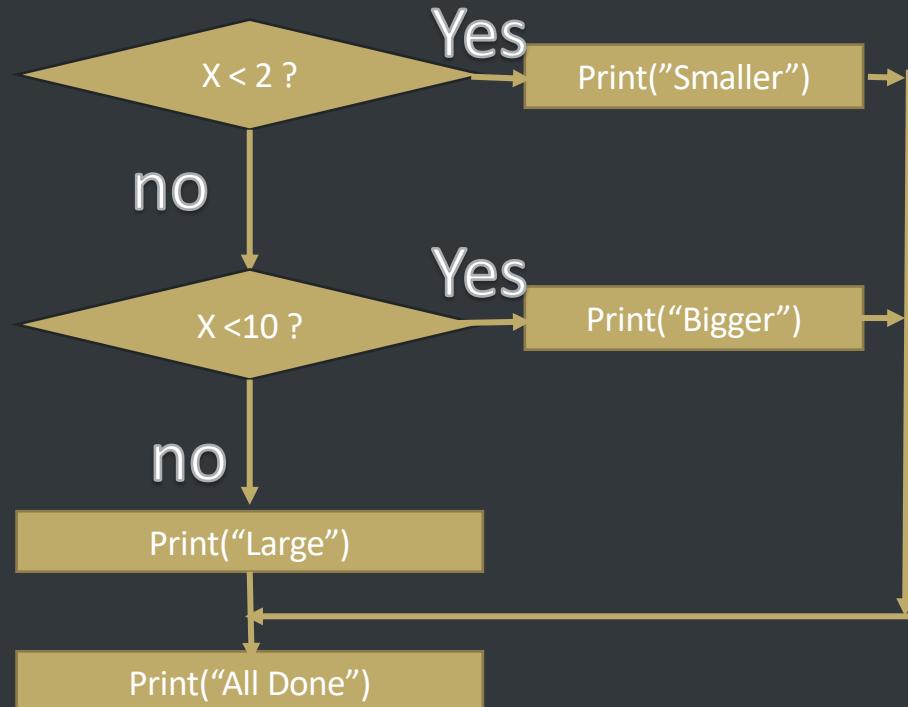
```
if x < 2 :  
    print("Small")  
elif x < 10 :  
    print("Medium")  
else:  
    print("Large")  
print("All Done")
```

WHEN X = 5



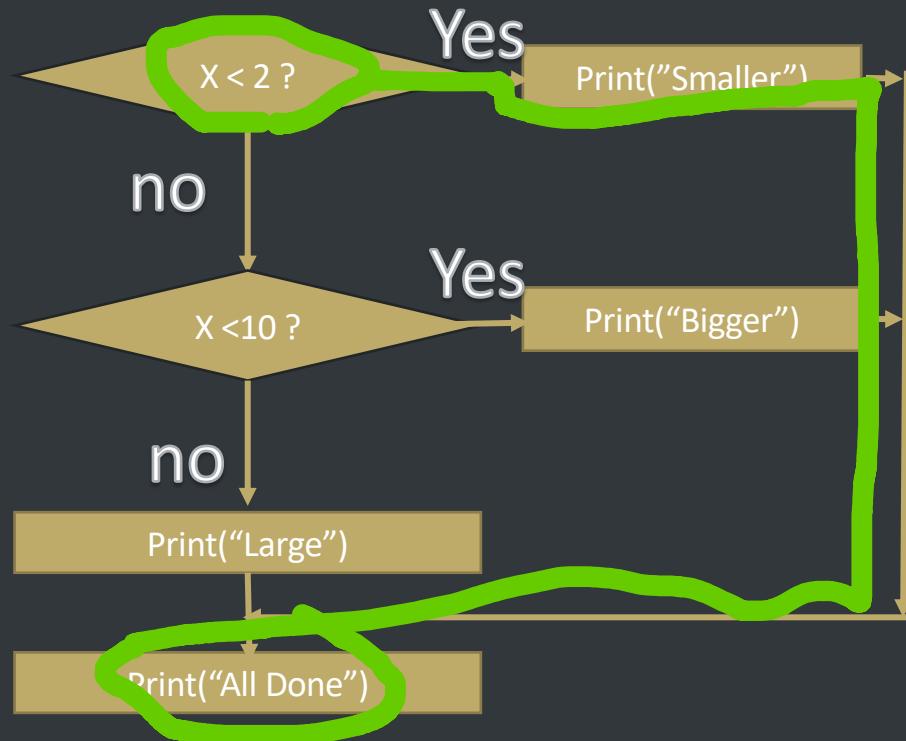
WHEN X = 1

```
if x < 2 :  
    print("Small")  
elif x < 10 :  
    print("Medium")  
else:  
    print("Large")  
print("All Done")
```



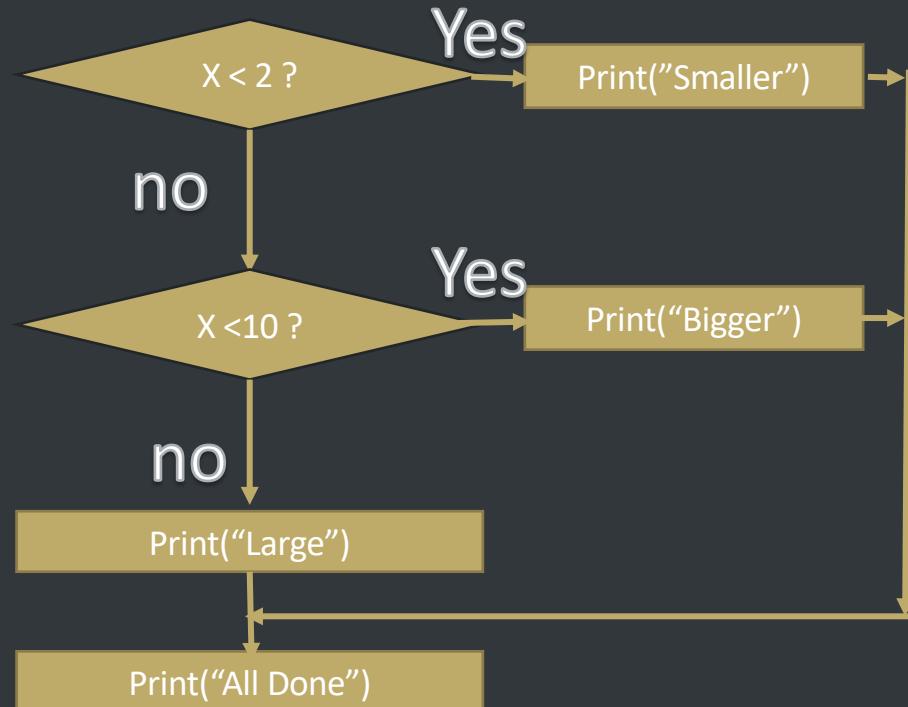
WHEN X = 1

```
if x < 2 :  
    print("Small")  
elif x < 10 :  
    print("Medium")  
else:  
    print("Large")  
print("All Done")
```



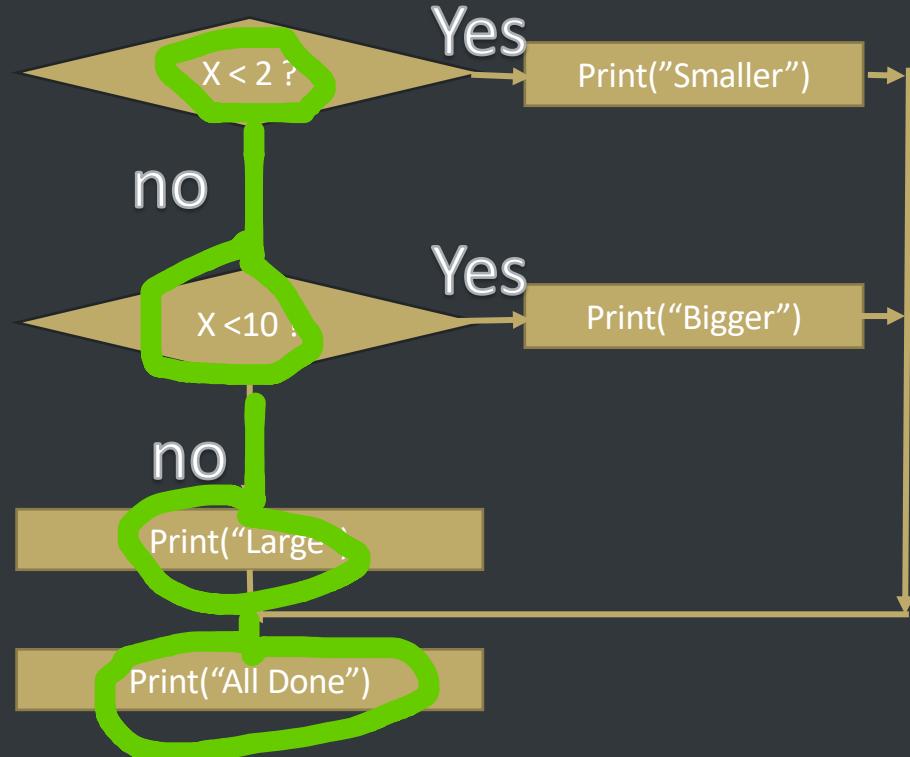
WHEN X = 20

```
if x < 2 :  
    print("Small")  
elif x < 10 :  
    print("Medium")  
else:  
    print("Large")  
print("All Done")
```



WHEN X = 20

```
if x < 2 :  
    print("Small")  
elif x < 10 :  
    print("Medium")  
else:  
    print("Large")  
print("All Done")
```



MULTI-WAY

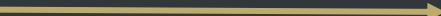
```
#No else
x = 5
if x < 2 :
    print("Small")
elif x < 10 :
    print("Medium")
print("All Done")
```

NO ELSE MEANS THAT YOU MIGHT NOT CATCH ALL THE POSSIBILITIES

MULTI-WAY

```
#No else
x = 5
if x < 2 :
    print("Small")
elif x < 10 :
    print("Medium")
print("All Done")
```

NO ELSE MEANS THAT YOU MIGHT NOT CATCH ALL THE POSSIBILITIES

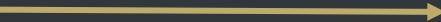


Medium
All Done

MULTI-WAY

```
#No else
x = 5
if x < 2 :
    print("Small")
elif x < 10 :
    print("Medium")
print("All Done")
```

NO ELSE MEANS THAT YOU MIGHT NOT CATCH ALL THE POSSIBILITIES



Medium
All Done

MULTI-WAY

```
#No else
x = 20
if x < 2 :
    print("Small")
elif x < 10 :
    print("Medium")
print("All Done")
```

NO ELSE MEANS THAT YOU MIGHT NOT CATCH ALL THE POSSIBILITIES

All Done

MULTI-WAY (ORDER)

```
if x < 2 :
    print("Small")
elif x < 10 :
    print("Medium")
elif x < 20 :
    print("Big")
elif x < 40 :
    print("Large")
elif x < 100 :
    print("Huge")
else :
    print("Ginormous")
```

TRY / EXCEPT

Surround a dangerous section of code with `try` and `except`

If the code in `try` works – the `except` is skipped

If the code in the `try` fail – it jumps the `except` section

TRY / EXCEPT

Surround a dangerous section of code with `try` and `except`

If the code in `try` works – the `except` is skipped

If the code in the `try` fail – it jumps the `except` section

```
# Cat notry.py
astr = "Hello Bob!"
istr = int(astr)
print("First", istr)
astr = "123"
instr = int(astr)
print("Second", istr)
```

TRY / EXCEPT

Surround a dangerous section of code with `try` and `except`

If the code in `try` works – the `except` is skipped

If the code in the `try` fail – it jumps the `except` section

```
# Cat notry.py
astr = "Hello Bob!"
istr = int(astr)
print("First", istr)
astr = "123"
instr = int(astr)
print("Second", istr)
```

We cannot turn a string that doesn't contain
numbers into a integer

TRY / EXCEPT

Surround a dangerous section of code with `try` and `except`

If the code in `try` works – the `except` is skipped

If the code in the `try` fail – it jumps the `except` section

```
# Cat notry.py
astr = "Hello Bob!"
istr = int(astr)
print("First", istr)
astr = "123"
instr = int(astr)
print("Second", istr)
```

```
Traceback (most recent call last):
  File "/Users/sohnshine/Documents/o.py", line 3, in <module>
    istr = int(astr)
ValueError: invalid literal for int() with
base 10: 'Hello Bob!'
```

TRY / EXCEPT

```
# python tryexcept.py
astr = "Hello Bob!"
try:
    istr = int(astr)
except:
    istr = -1

print("First", istr)

astr = "123"
try:
    istr = int(astr)
except:
    istr = -1

print("Second", istr)
```

When the first conversion fails, it just drops into the except: clause and the program continues

TRY / EXCEPT

```
# python tryexcept.py
astr = "Hello Bob!"
try:
    istr = int(astr)
except:
    istr = -1

print("First", istr)

astr = "123"
try:
    istr = int(astr)
except:
    istr = -1

print("Second", istr)
```

When the first conversion fails, it just drops into the except: clause and the program continues

TRY / EXCEPT

```
# python tryexcept.py
astr = "Hello Bob!"
try:
    istr = int(astr)
except:
    istr = -1

print("First", istr)
```

When the first conversion fails, it just drops into the except: clause and the program continues

```
astr = 123
try:
    istr = int(astr)
except:
    istr = -1

print("Second", istr)
```

When the second conversion succeeds - it just skips the except: clause and the program continues

TRY / EXCEPT

```
# python tryexcept.py
astr = "Hello Bob!"    rawstr = input("Enter a number:")
try:                      try:
    istr = int(astr)    ival = int(rawstr)
except:                  except:
    istr = -1           ival = -1
print("First", istr)    if ival > 0 :
                        print("Nice Work!")
else:
                        print("Not a number")
astr = "123"
try:
    istr = int(astr)
except:
    istr = -1
print("Second", istr)
```

, it just
and the

First -1
Second 123

When the second conversion succeeds - it
just skips the except: clause and the
program continues

TRY / EXCEPT

```
rawstr = input("Enter a number:")
try:
    ival = int(rawstr)
except:
    ival = -1

if ival > 0 :
    print("Nice Work!")
else:
    print("Not a number")
```

TRY / EXCEPT

```
rawstr = input("Enter a number:")
try:
    ival = int(rawstr)
except:
    ival = -1

if ival > 0 :
    print("Nice Work!")
else:
    print("Not a number")
```

Enter a number:3
Nice Work!

Enter a number:three
Not a number

SUMMARY



COMPARISON OPERATORS

==	>
<=	<
>=	!=



INDENTATION

We use indentation like syntax in python.
Indentation shows what the block of code is
and shows which code belong where



NESTED DECISIONS

Nested decisions are integral part of indents.



ONE WAY DECISIONS

If statements – a conditional



TWO WAY DECISIONS

Involves if: and else: in the code



MULTI WAY DECISIONS

Using elif, there are multiple ways that the code can run through



TRY / EXCEPT TO COMPENSATE ERROR

In order to make sure there is no traceback error, we anticipate the error that can appear and compensate for it.

MINI CHALLENGE: BROKEN CODES

Find which line of code will never print regardless of value for x and
rewrite them so all lines are will run.

```
if x <2 :  
    print("Below 2")  
elif x >= 2 :  
    print("Two or more")  
else :  
    print("Something else")
```

```
if x < 2 :  
    print("Below 2")  
elif x < 20 :  
    print("Below 20")  
elif x < 10 :  
    print("Below 10")  
else:  
    print("Something else")
```

MINI CHALLENGE: BROKEN CODES

```
if x <2 :  
    print("Below 2")  
elif x >= 2 :  
    print("Two or more")  
else :  
    print("Something else")
```

All numbers are either smaller than 2
or equal / bigger than two

```
if x < 2 :  
    print("Below 2")  
elif x < 20 :  
    print("Below 20")  
elif x < 10 :  
    print("Below 10")  
else:  
    print("Something else")
```

All numbers below 20 are also below 10

MINI CHALLENGE: BROKEN CODES

```
if x <2 :  
    print("Below 2")  
elif x >= 2 :  
    print("Two or more")  
else :  
    print("Something else")
```

All numbers are either smaller than 2
or equal / bigger than two

```
if x < 2 :  
    print("Below 2")  
elif x < 20 :  
    print("Below 20")  
elif x < 10 :  
    print("Below 10")  
else:  
    print("Something else")
```

All numbers below 20 are also below 10

MINI CHALLENGE: BROKEN CODES

```
if x < 2 :  
    print("Below 2")  
elif x >= 2:  
    print("Two or more")
```

```
if x < 2:  
    print("Below 2")  
elif x < 10 :  
    print("Below 10")  
elif x < 20:  
    print("Below 10")  
else:  
    print("Something else")
```



DATUM

Yonsei Data Science Academia

FUNCTIONS

FUNCTION

There are two kinds of functions in python

- Built in function that are provided as part of python (e.g. `print()`, `input()`,
`float()`, `int()`....
 - Functions that we define ourselves and then use

We treat built in functions as new reserved words (avoid them in variable names)

FUNCTION

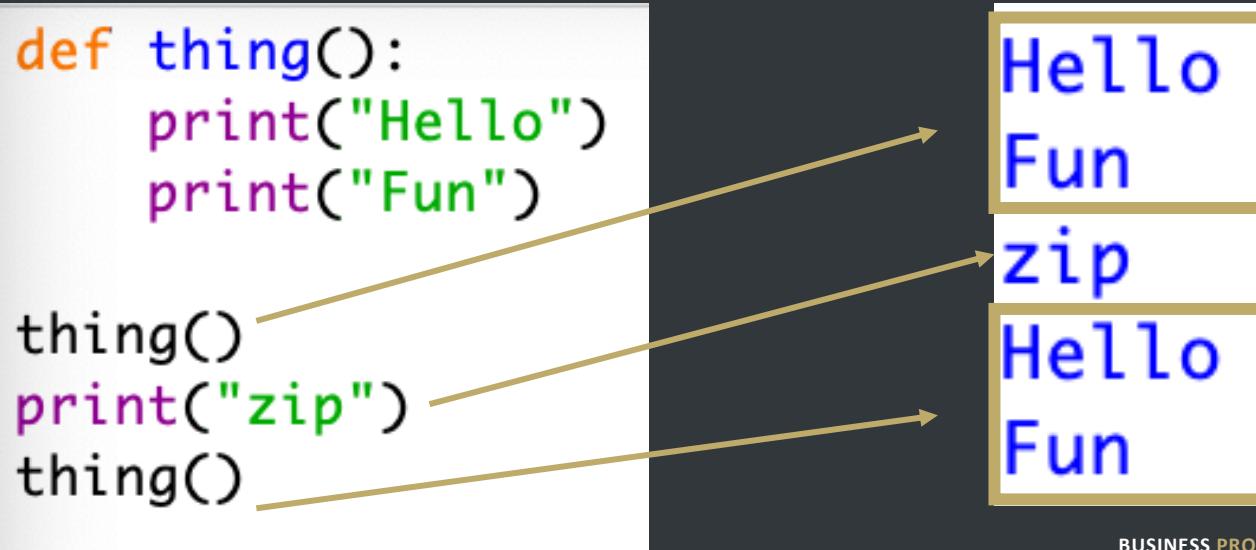
- Function is a reusable code that takes **arguments** as input and returns results
 - We write a **function** using **def** reserved word
 - We call/invoke the **function** by using the function name, parentheses, and arguments in an expression

STORE (AND REUSED) STEPS

Def stands for define function

```
def thing():
    print("Hello")
    print("Fun")
```

```
thing()
print("zip")
thing()
```



```
Hello
Fun
zip
Hello
Fun
```



FUNCTION

Max function find the largest “letter”
Min function finds the shortest “letter”

argument

```
big = max("Hello world")
```

Assignment

w

result

FUNCTION

Max function find the largest “letter”
Min function finds the shortest “letter”

argument

```
big = max("Hello world")
```

Assignment

w

result



MAX FUNCTION

A function is **some stored code** that we use.

A function takes some **input** and produces an **output**

HELLO WORLD
(A STRING)



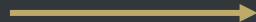
“W”
(A STRING)

MAX FUNCTION

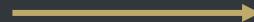
A function is **some stored code** that we use.

A function takes some **input** and produces an **output**

HELLO WORLD
(A STRING)



```
Def max(inp)
    Blah
    Blah
    Blah
    For x in y
```



“W”
(A STRING)

DEFINITION AND USES

Once we have **defined** a function, we can **call** (or **invoke**) it as any times as we like

This is the **store and reuse** pattern

```
x = 5
print("Hello")

def print_rhyme():
    print("How many wood can a woodchuck chi")
    print("If woodchuck could chuck wood")

print("Yo")
print_rhyme()
x = x + 2
print(x)
```

DEFINITION AND USES

Once we have **defined** a function, we can **call** (or **invoke**) it as many times as we like

This is the **store and reuse** pattern

```
x = 5
print("Hello")

def print_rhyme():
    print("How many wood can a woodchuck chuck")
    print("If woodchuck could chuck wood")

print("Yo")
print_rhyme()
x = x + 2
print(x)
```

```
Hello
Yo
How many wood can a woodchuck chuck
If woodchuck could chuck wood
7
```

ARGUMENTS

An **argument** is a value we pass through the **function** as its **input**
when we call the function

We use **arguments** so we can direct **function** to do different kinds
of work when we call it at **different times**

We put the **arguments** in the parentheses after the **name** of the
functions

argument

```
big = max("Hello world")
```

PARAMETER

A parameter is a variable which we use **in** the function **definition**. It is a “handle” that allows the code in the function to access the **arguments** for a particular function invocation.

```
def greet(lang):  
    if lang == 'es':  
        print("Hola!")  
    elif lang == 'fr':  
        print("Bonjour")  
    else:  
        print("Hello")  
  
greet('en')  
greet('es')  
greet('fr')
```

Hello
Hola!
Bonjour

RETURN VALUES

Often a function will take its arguments, do some computation and **return** a value to be used as the value of the function call in the **calling expression**. The **return** keyword is used for this.

Functions that return values are “fruitful” because they produce a **result**.

The **return** statement ends the **function** execution and “sends back” the **result** of the **function**.

I DON'T UNDERSTAND

Don't worry! You
will later on



DATUM

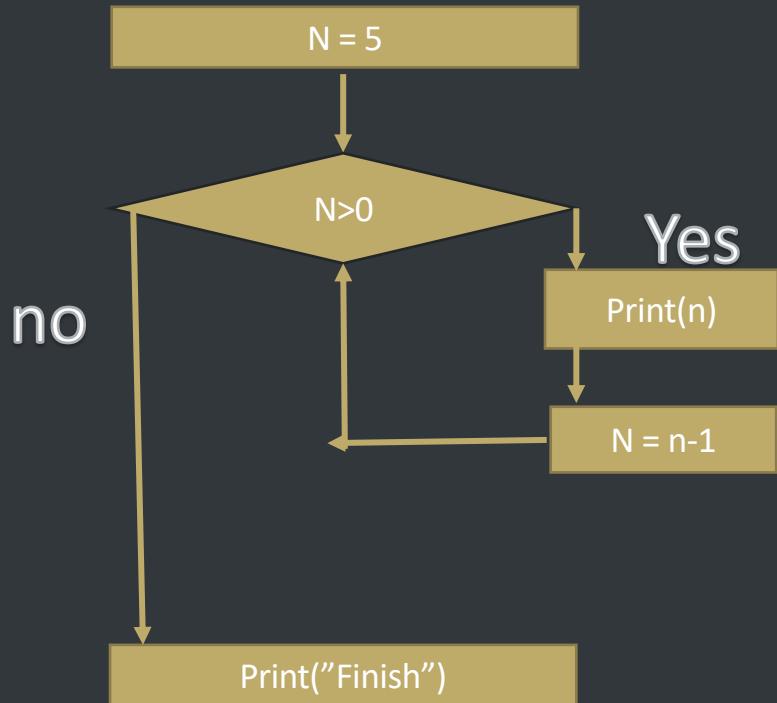
Yonsei Data Science Academia

LOOPS AND ITERATION

LOOP

Loops have **iteration** variables that change each time through a loop. Often these iteration variables go through a sequence of numbers

```
n = 5
while n > 0 :
    print(n)
    n = n - 1
print("Blastoff!")
print(n)
```



```
n = 5
while n > 0 :
    print(n)
    n = n - 1
print("Blastoff!")
print(n)
```

WHILE

```
5
4
3
2
1
Blastoff!
0
```

```
n = 5
while n > 0 :
    print(n)
    n = n-1
print("Blastoff!")
print(n)
```

MINI - CHALLENGE

FIX THIS CODE AND DRAW A DIAGRAM EXPLAINING THE CODE

```
n = 5
while n > 0
    print("Lather")
    print("Rinse")
print("Dry Off!")
```

BREAKING OUT OF A LOOP

Break statement ends the current loop and jumps to the statement immediately following the loop

```
while True:  
    line = input("> ")  
    if line == 'done' :  
        break  
    print(line)  
print("Done!")
```

BREAKING OUT OF A LOOP

Break statement ends the current loop and jumps to the statement immediately following the loop

```
while True:  
    line = input("> ")  
    if line == 'done' :  
        break  
    print(line)  
print("Done!")
```

```
> this is going to  
this is going to  
> repeat whatever i type  
repeat whatever i type  
> unless i type  
unless i type  
> done  
Done!
```

BREAKING OUT OF A LOOP

Break statement ends the current loop and jumps to the statement immediately following the loop

```
while True:  
    line = input("> ")  
    if line == 'done' :  
        break  
    print(line)  
print("Done!")
```

```
> this is going to  
this is going to  
> repeat whatever i type  
repeat whatever i type  
> unless i type  
unless i type  
> done  
Done!
```

FINISH ITERATION WITH CONTINUE

The continue statement ends the current iteration and jumps to the top of the loop and starts the next iteration

```
while True:
    line = input("> ")
    if line[0] == '#':
        continue
    if line == 'done':
        break
    print(line)
print("Done!")
```

FINISH ITERATION WITH CONTINUE

The continue statement ends the current iteration and jumps to the top of the loop and starts the next iteration

```
while True:  
    line = input("> ")  
    if line[0] == '#':  
        continue  
    if line == 'done':  
        break  
    print(line)  
print("Done!")
```

```
> this code will repeat me  
this code will repeat me  
> # unless i put a hashtag  
> # it'll stay slient when i do  
> and this will end when i type  
and this will end when i type  
> done  
Done!
```

INDEFINITE LOOP

While loops are called “indefinite loops” because they keep going until a logical condition becomes false.

The loops we have seen so far are pretty easy to examine to see if they will terminate or if they will be “infinite loops”

Sometimes it is a little harder to be sure if a loop will terminate

DEFINITE LOOP

We have list of items of the lines in the file- effectively a **finite set** of things

We can write a loop to run the loop once **for** each of the items in a set using the **python for** construct

These loops are called “**definite loops**” because they execute an exact number of times

We say that “**definite loops iterate through the members of the set**”

A SIMPLE DEFINITE LOOP

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

```
5  
4  
3  
2  
1  
Blastoff!
```

A SIMPLE DEFINITE LOOP

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

5
4
3
2
1
Blastoff!

```
friends = ['Joseph', 'Glenn', 'Sally']  
for friend in friends :  
    print('Happy New Year:', friend)  
print("Done!")
```

Happy New Year: Joseph
Happy New Year: Glenn
Happy New Year: Sally
Done!

CHALLENGE

WRITE A CODE THAT FINDS THE LARGEST VALUE IN THE LIST OF NUMBERS:

[9, 41, 12, 3, 74, 15]

Hint:

You would need to set a variable that you'll continue to compare the numbers to (i.e. `largest_so_far = -1`)

before -1
9 9
41 41
41 12
41 3
74 74
74 15
After 74

This is my output

CHALLENGE

WRITE A CODE THAT FINDS THE LARGEST VALUE IN THE LIST OF NUMBERS:

[9, 41, 12, 3, 74, 15]

Hint:

You would need to set a variable that you'll continue to compare the numbers to (i.e. `largest_so_far = -1`)

Second hint!

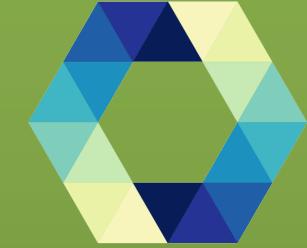
```
if the_num > largest_so_far :  
    largest_so_far = the_num
```

CHALLENGE ANSWERS

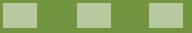
WRITE A CODE THAT FINDS THE LARGEST VALUE IN THE LIST OF NUMBERS:

[9, 41, 12, 3, 74, 15]

```
1  largest_so_far = -1
2  print("before", largest_so_far)
3  for the_num in [9, 41, 12, 3, 74, 15] :
4      if the_num > largest_so_far :
5          largest_so_far = the_num
6      print(largest_so_far, the_num)
7
8  print("After", largest_so_far)
```



DATUM
Yonsei Data Science Academia



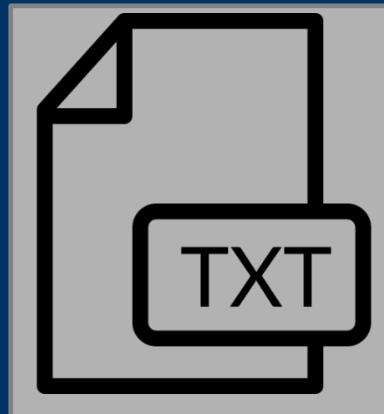
Lesson 7

Files

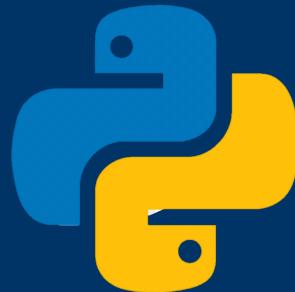
Opening a file



.xlsx
.csv
.tsv



.txt



.py

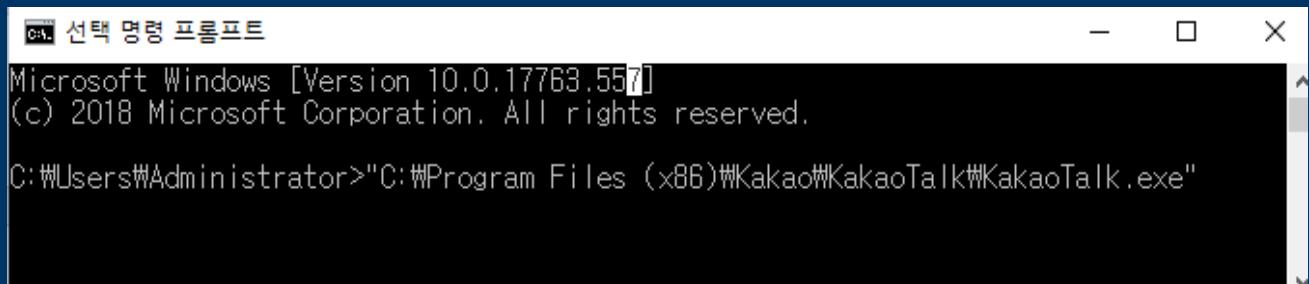
- Python can locate a certain file with the open function
 - Read
 - Write
 - Append
 - Close
 - Etc.

File handler

- File handler is used to do something with a file within python

handler = `open(file, mode)`

1. Name of the function
2. Function to open a certain file
3. Name/location of a file
4. Mode of the open function

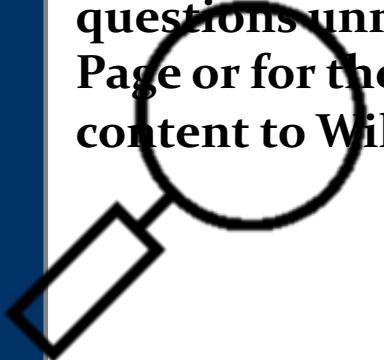


Modes of open()

handler = **open(file, mode)**

- “r” = for reading only
- “w” = for overwriting a file
- “r+” = for reading and writing
- “a” = for appending to a file

Welcome! This page is only for discussing the contents of the [Main Page](#). It isn't for general questions unrelated to the Main Page or for the addition of content to Wikipedia articles.



Modes of open()

handler = **open(file, mode)**

- “r” = for reading only
- “w” = for overwriting a file
- “r+” = for reading and writing
- “a” = for appending to a file

Welcome! This page is only for discussing the contents of the [Main Page](#). It isn't for general questions unrelated to the Main Page or for the addition of content to Wikipedia articles.

Modes of open()

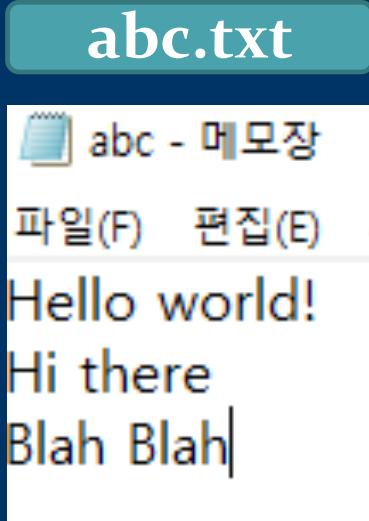
handler = **open(file, mode)**

- “r” = for reading only
- “w” = for overwriting a file
- “r+” = for reading and writing
- “a” = for appending to a file

Welcome! This page is only for discussing the contents of the [Main Page](#). It isn't for general questions unrelated to the Main Page or for the addition of content to Wikipedia articles.

File handle sequence

abc.txt



```
abc - 메모장
파일(F) 편집(E)
Hello world!
Hi there
Blah Blah|
```



File handle sequence

```
file = open("abc.txt")
for sth in file:
    print(sth)
```

result

```
=====
Hello world!
Hi there
Blah Blah
>>> |
```

- File handle sequence
 - Reads each line of file

For **█** in **█** = for each line in **█**

File handle sequence

Counting the number of lines in a file

```
file = open("abc.txt")  
count = 0  
for sth in file:  
    count+= 1  
print("There are", count, "lines")
```

Opening a file

Creating a variable 'count'

For each line in 'abc.txt'

Add 1 to count

Printing the result

#newline characters

- In python, we express a change of line with ‘\n’
 - Ex)

Gallimimus was a theropod dinosaur that lived in what is now Mongolia about 70 million years ago, during the Late Cretaceous.



What you see on the web

How python sees the same text



Gallimimus was a theropod dinosaur that\n lived in what is now Mongolia about 70\n million years ago, during the Late\n Cretaceous.

#newline characters

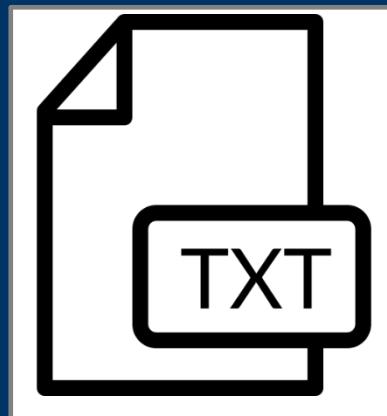
- This is applied within python as well

```
text = "Ask questi#n#ons about us#n#ing Wikipedia"  
print(text)
```

```
Ask questi  
ons about us  
ing Wikipedia  
>>> |
```

Reading a file

- `str_file = open('abcd.txt').read()`
 - `.read()` : function that converts whatever inside a txt file into a string



`str_file = "whatever was inside the text file"`

Counting the number of words in a text

- File handler : lets us do something to a file
- `len(x)`: returns the length of x
- Can we count how many words there are in a text file?
 - We will be reading “wiki.txt”
 - `len()` only works on **strings**

- 1. open the file
- 2. convert the file into a string
- 3. get the length of the string

Counting the number of words in a text

```
file = open("wiki.txt")
str_file = file.read()
print(len(str_file))
```

File handler

Converting file with read()

Printing the length of string
= How many words are in the file?

Extracting only what we want

- Remember the functions we learned in ‘strings’?
 - `line.startswith()`
 - `line.endswith()`
- How can we get the lines with the information we want?

Extracting only what we want

- Situation: from ‘wiki.txt’ , we only want the lines that start with ‘From’
- How can we make python print the lines that start with ‘From’?
- How can we print lines that **do not** start with ‘From’?

Extracting only what we want

- **wiki.txt**

From 1 pm we will start our session with the basics of python
lunch won't be provided so you will have to come after having lunch
From Chapter 1 to Chapter 7, we will learn the basic algorithms
these chapters provide information about basic methods to solve problems using python
this makes us easily solve problems that will be boring if we did it ourselves, taking way too much time
From 8 pm, we will have dinner.
The next day, we will start our sessions quite early
From Chapter 8, we deal with data structures and organizing data.

- 1. open the file! (wiki.txt)
- 2. for each line in the txt file
- 3. if the line startswith “From”
- 4. print

Extracting only what we want

```
file = open('wiki.txt')  
  
for line in file:  
  
    if line.startswith('From'):  
  
        print(line)
```

```
file = open('wiki.txt')  
  
for line in file:  
  
    if not line.startswith('From'):  
  
        print(line)
```

The background features a decorative graphic on the left side. It consists of a vertical column of thin, light-colored vertical lines. A diagonal line cuts across the graphic, starting from the bottom-left corner and extending upwards towards the top-right. The area below the diagonal is filled with a solid orange color, while the area above it is a lighter yellow. The rest of the slide has a solid dark blue background.

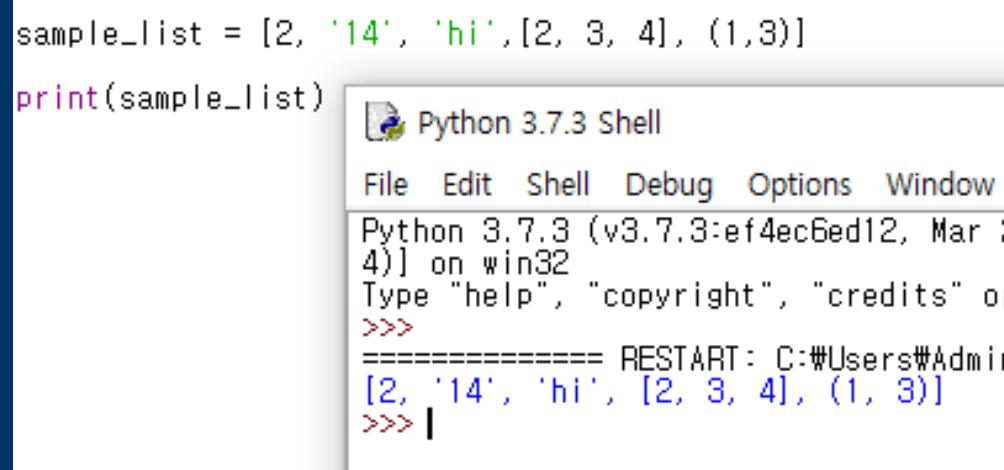
Lesson 8

Lists

What's a list?

- A group of **values** put inside a single **variable**
 - Lets us call in a load of values with a single variable
 - Makes the code more concise, simple, easier
 - Format: `list1 = [a, b, c]`

- Things that can go inside a list
 - strings
 - numbers
 - lists
 - *tuples
 - etc



```
sample_list = [2, '14', 'hi', [2, 3, 4], (1,3)]
print(sample_list)
```

Python 3.7.3 Shell

File Edit Shell Debug Options Window

Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 24) on win32

Type "help", "copyright", "credits" or

>>>

===== RESTART: C:\Users\Admin

[2, '14', 'hi', [2, 3, 4], (1, 3)]

>>> |

When is it used?

- List is commonly used with loops
 - List: stores multiple values
 - Loops: suitable for processing multiple values

```
friends = ["Joseph", "Glenn", "Sally", "Kevin", "Sean", "Jane"]
for friend in friends:
    print('Happy new year', friend)
print('done!')
```

Try

- We have a list of 5 numbers, 19, 23, 16, 27, 84
- How can we create a list where each variable is five times larger than the original list?

When is it used?

In [8]:

```
number1 = 10
number2 = 192
number3 = -12
number4 = 151
number5 = 242
number6 = 13
number7 = 23
largest = number1
if largest < number2:
    largest = number2
if largest < number3:
    largest = number3
if largest < number4:
    largest = number4
if largest < number5:
    largest = number5
if largest < number6:
    largest = number6
if largest < number7:
    largest = number7
print(largest)
```

In [22]:

```
largest = None
numbers = [10, 192, -12, 151, 242, 13, 23]
for value in numbers:
    if largest == None:
        largest = value
    elif value > largest:
        largest = value
print(largest)
```



Strings vs Lists

String

- immutable
 - cannot fix
 - `len()` = number of characters
- ```
string = "hello"
len(string) = 5
```

## List

- mutable
  - can fix wherever you want
  - `len()` = number of values
- ```
list = [1, 2, 'hi', [2, 4, 5]]
len(list) = 4
```

Fixing a list

- Fixing a string

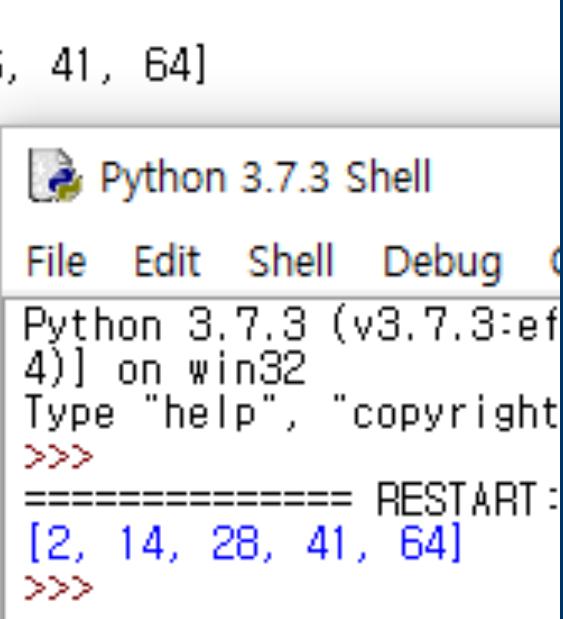
```
fruit = 'Banana'  
fruit[0] = 'b'
```

```
Type "help", "copyright", "credits" or "license()" for more information  
>>>  
===== RESTART: C:\Users\Administrator\Desktop\temp.py =====  
Traceback (most recent call last):  
  File "C:\Users\Administrator\Desktop\temp.py", line 4,  
    fruit[0] = 'b'  
TypeError: 'str' object does not support item assignment  
>>> |
```

Fixing a list

- Fixing a list

```
lotto = [2, 14, 26, 41, 64]
lotto[2] = 28
print(lotto)
```



The image shows a screenshot of the Python 3.7.3 Shell. The code in the left pane is:

```
lotto = [2, 14, 26, 41, 64]
lotto[2] = 28
print(lotto)
```

The output in the right pane is:

```
Python 3.7.3 (v3.7.3:ef4) on win32
Type "help", "copyright"
>>>
=====
RESTART:
[2, 14, 28, 41, 64]
>>>
```

The Range Function

- returns a list of numbers from 0 to a parameter
 - `range(4) == [0, 1, 2, 3]`
- used with `len()`

```
>>> friends = ["Joseph", "Glenn", "Sally", "Kevin", "Sean", "Jane"]
>>> print(len(friends))
6
>>> print(range(len(friends)))
range(0, 6)
>>> |
```

Diverse Methods for Solving a Problem

```
friends = ["Joseph", "Glenn", "Sally", "Kevin", "Sean", "Jane"]
for friend in friends:
    print('Happy new year', friend)
```



```
friends = ["Joseph", "Glenn", "Sally", "Kevin", "Sean", "Jane"]
for i in range(len(friends)):
    friend = friends[i]
    print('Happy new year', friend)
```

```
Happy new year Joseph
Happy new year Glenn
Happy new year Sally
Happy new year Kevin
Happy new year Sean
Happy new year Jane
>>> |
```

Concatenating/Slicing a list

Concatenating

```
>>> a = [1,2,3]
>>> b = [4,5,6]
>>> c = a+b
>>> print(c)
[1, 2, 3, 4, 5, 6]
>>>
```

Slicing

```
>>> t = [9, 141, 12, 3, 74, 15]
>>> t[1:3]
[141, 12]
>>> t[:4]
[9, 141, 12, 3]
>>> t[3:]
[3, 74, 15]
>>> t[:]
[9, 141, 12, 3, 74, 15]
>>> t[:-2]
[9, 141, 12, 3]
>>> |
```

t[a:b] =
from a to b

Methods used in Lists

- Append
- Count
- Extend
- Index
- Sort
- Insert
- etc

Creating a List

- We can create an empty list and add elements using the **append** method
- The appended elements go to the end of the list

```
stuff = list()
stuff.append("book")
stuff.append(91)
stuff.append("snacks")

print(stuff)
```

Checking if in List

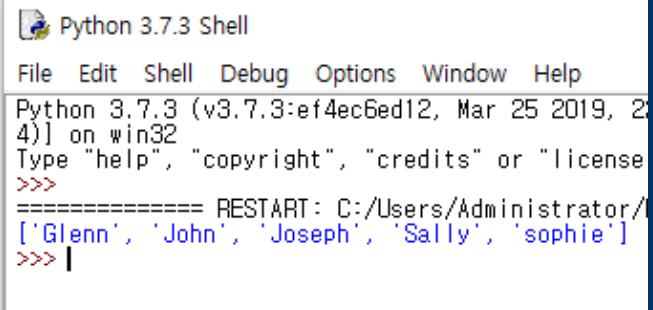
- To check if a certain variable is in a list, we use **in**

```
>>> numbers = [1,6,73,26,45,2]
>>> 9 in numbers
False
>>> 1 in numbers
True
>>> |
```

Sorting Variables

- We can rearrange the order of variables
 - If all variables are the same type
 - In alphabetical / numerical order
 - Capital > Lower case

```
friends = ['Joseph', 'sophie', 'John', 'Glenn', 'Sally']
friends.sort()
print(friends)
```



Python 3.7.3 Shell

File Edit Shell Debug Options Window Help

Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 22:22:05) [MSC v.1916 64-bit (AMD64)] on win32

Type "help", "copyright", "credits" or "license()" for more information

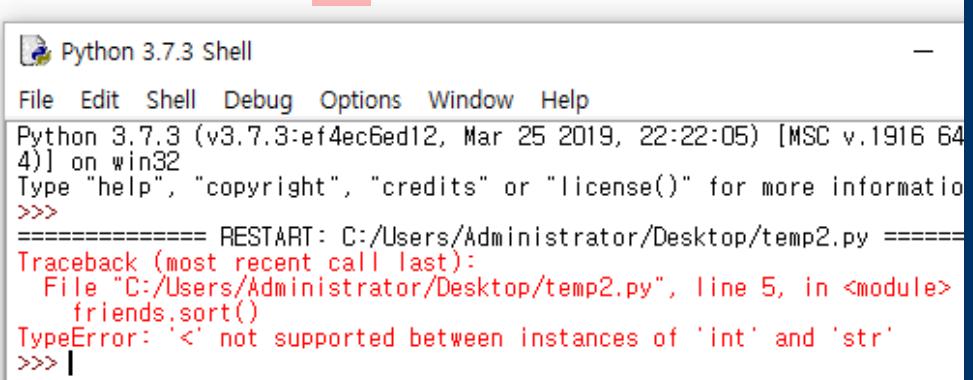
>>>

===== RESTART: C:/Users/Administrator/Desktop/temp2.py =====

['Glenn', 'John', 'Joseph', 'sophie', 'Sally']

>>> |

```
friends = ['Joseph', 'sophie', 'John', 19, 'Glenn', 'Sally']
friends.sort()
print(friends)
```



Python 3.7.3 Shell

File Edit Shell Debug Options Window Help

Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 22:22:05) [MSC v.1916 64-bit (AMD64)] on win32

Type "help", "copyright", "credits" or "license()" for more information

>>>

===== RESTART: C:/Users/Administrator/Desktop/temp2.py =====

Traceback (most recent call last):

File "C:/Users/Administrator/Desktop/temp2.py", line 5, in <module>

friends.sort()

TypeError: '<' not supported between instances of 'int' and 'str'

>>> |

Try

- Try arranging the given lists, see what happens
 - `list1 = ['Gray', 'Purple', 'red', 'black', 'Yellow', 'green']`
 - `[4, 'hi', 15, 'ag']`

Built-in Functions

- `nums = [3, 41, 12, 8, 53, 13]`
 - Try:
 - `len(nums)` > number of variables
 - `max(nums)` > the largest number
 - `min(nums)` > the smallest number
 - `sum(nums)` > sum of numbers

Getting the Average 1

- Using all that we learned, we can make a program that gets the average

```
total = 0
count = 0
while True:
    inp = input("Enter:")
    if inp == 'done' :
        break
    val = float(inp)
    total+=val
    count+=1
av = total/count
print(av)
```

Getting the Average 2

- Let's make a same program using a list
 - `len(list)`
 - `sum(list)`

```
total = 0
count = 0
while True:
    inp = input("Enter:")
    if inp == 'done' :
        break
    val = float(inp)
    total+=val
    count+=1
av = total/count
print(av)
```

Getting the Average 2

```
numlist = list()  
while True:
```

```
    print(average)
```

Strings & Lists

- Strings and Lists are often used together

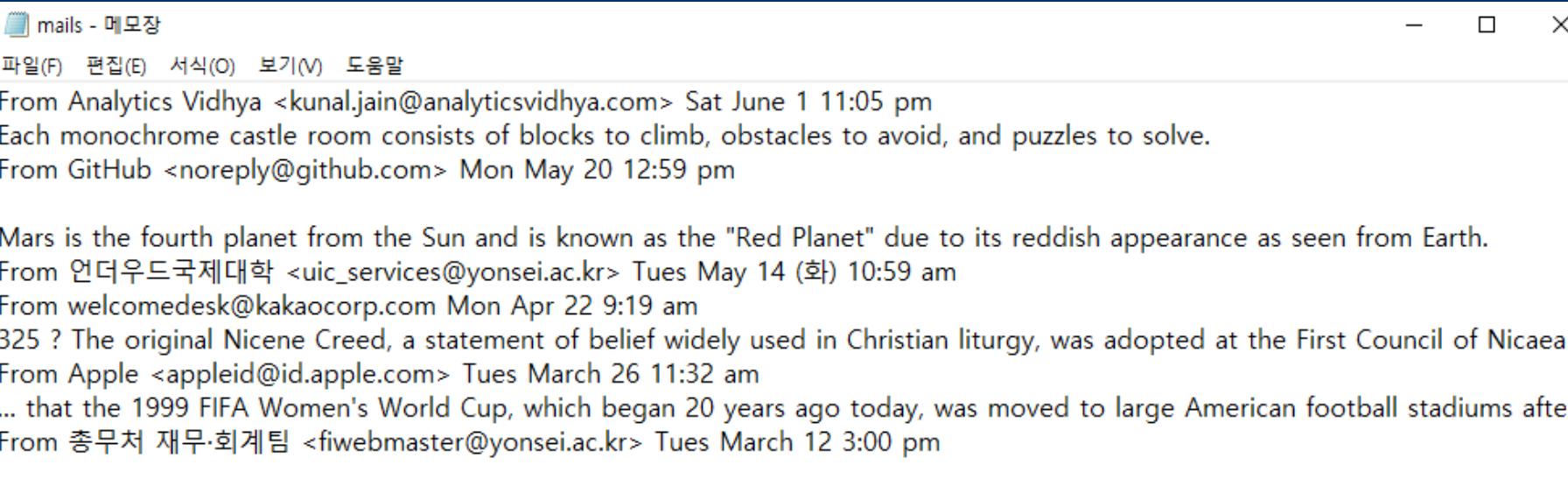
```
>>> abc = ('These three words')
>>> stuff = abc.split()
>>> print(stuff)
['These', 'three', 'words']
>>> print(len(stuff))
3
>>> print(stuff[0])
These
>>> |
```

```
>>> line = 'A lot of spaces'
>>> etc = line.split()
>>> print(etc)
['A', 'lot', 'of', 'spaces']
>>>
>>> line = 'first;second;third'
>>> thing = line.split()
>>> print(thing)
['first;second;third']
>>>
>>> thinng = line.split(';')
>>> print(thinng)
['first', 'second', 'third']
>>> |
```

String parameters of split() : indicates where the string should be sliced

Exercise

- Let's read the file mails.txt
- Remove all junk lines
- Make each lines of email into a list
- See from which group the mails are coming from



mails - 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말

From Analytics Vidhya <kunal.jain@analyticsvidhya.com> Sat June 1 11:05 pm
Each monochrome castle room consists of blocks to climb, obstacles to avoid, and puzzles to solve.

From GitHub <noreply@github.com> Mon May 20 12:59 pm

Mars is the fourth planet from the Sun and is known as the "Red Planet" due to its reddish appearance as seen from Earth.

From 언더우드국제대학 <uic_services@yonsei.ac.kr> Tues May 14 (화) 10:59 am

From welcomedesk@kakaocorp.com Mon Apr 22 9:19 am

325 ? The original Nicene Creed, a statement of belief widely used in Christian liturgy, was adopted at the First Council of Nicaea

From Apple <appleid@id.apple.com> Tues March 26 11:32 am

... that the 1999 FIFA Women's World Cup, which began 20 years ago today, was moved to large American football stadiums after

From 총무처 재무·회계팀 <fiwebmaster@yonsei.ac.kr> Tues March 12 3:00 pm

```
line = "From <mw116208@yonsei.ac.kr> Sat June 22 12:04 pm"
words = line.split()
print(words)
```

```
['From', '<mw116208@yonsei.ac.kr>', 'Sat', 'June', '22', '12:04', 'pm']
>>> |
```

```
line = "From <mw116208@yonsei.ac.kr> Sat June 22 12:04 pm"
words = line.split()
print(words)

email = words[1]
print(email)
```

Python 3.7.3 Shell

File Edit Shell Debug Options Window Help

```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 22:22:05) [MSC v.1916 64 bit (AMD64)] on win32
```

```
Type "help", "copyright", "credits" or "license()" for more information.
```

```
>>>
```

```
===== RESTART: C:\Users\Administrator\Desktop\bootcamp\Chapter8\ver1.py =====
['From', '<mw116208@yonsei.ac.kr>', 'Sat', 'June', '22', '12:04', 'pm']
<mw116208@yonsei.ac.kr>
>>> |
```

```
fhand = open("C:\Users\Administrator\Desktop\bootcamp\Chapter8\mails.txt")
for line in fhand:
    line = line.rstrip()
    if not line.startswith('From '):
        continue
    words = line.split()
    for word in words:
        adr = list()
        if word.endswith(">com"):
            adr.append(word)
        elif word.endswith(">kr"):
            adr.append(word)
        else:
            continue
```

```
fhand = open("C:\Users\Administrator\Desktop\bootcamp\Chapter8\mails.txt")
for line in fhand:
    line = line.rstrip()
    if not line.startswith('From '):
        continue
    words = line.split()
    for word in words:
        adr = list()
        if word.endswith(">com"):
            adr.append(word)
        elif word.endswith(">kr"):
            adr.append(word)
        else:
            continue
```

```
fhand = open("C:\Users\Administrator\Desktop\bootcamp\Chapter8\mails.txt")
for line in fhand:
    line = line.rstrip()
    if not line.startswith('From '):
        continue
    words = line.split()
    for word in words:
        adr = list()
        if word.endswith("<com>"):
            adr.append(word)
        elif word.endswith("<kr>"):
            adr.append(word)
        else:
            continue
        print(adr)
```

```
fhand = open("C:\Users\Administrator\Desktop\bootcamp\Chapter8\mails.txt")
for line in fhand:
    line = line.rstrip()
    if not line.startswith('From '):
        continue
```

```
fhand = open("C:\\Users\\Administrator\\Desktop\\bootcamp\\Chapter8\\mails.txt")
for line in fhand:
    line = line.rstrip()
    if not line.startswith('From '):
        continue
    words = line.split()
    for word in words:
        adr = list()
        if word.endswith(">com"):
            adr.append(word)
        elif word.endswith(">kr"):
            adr.append(word)
        else:
            continue
        print(adr)
```

```
words = line.split()

for word in words:
    adr = list()

    if word.endswith(">com"):
        adr.append(word)

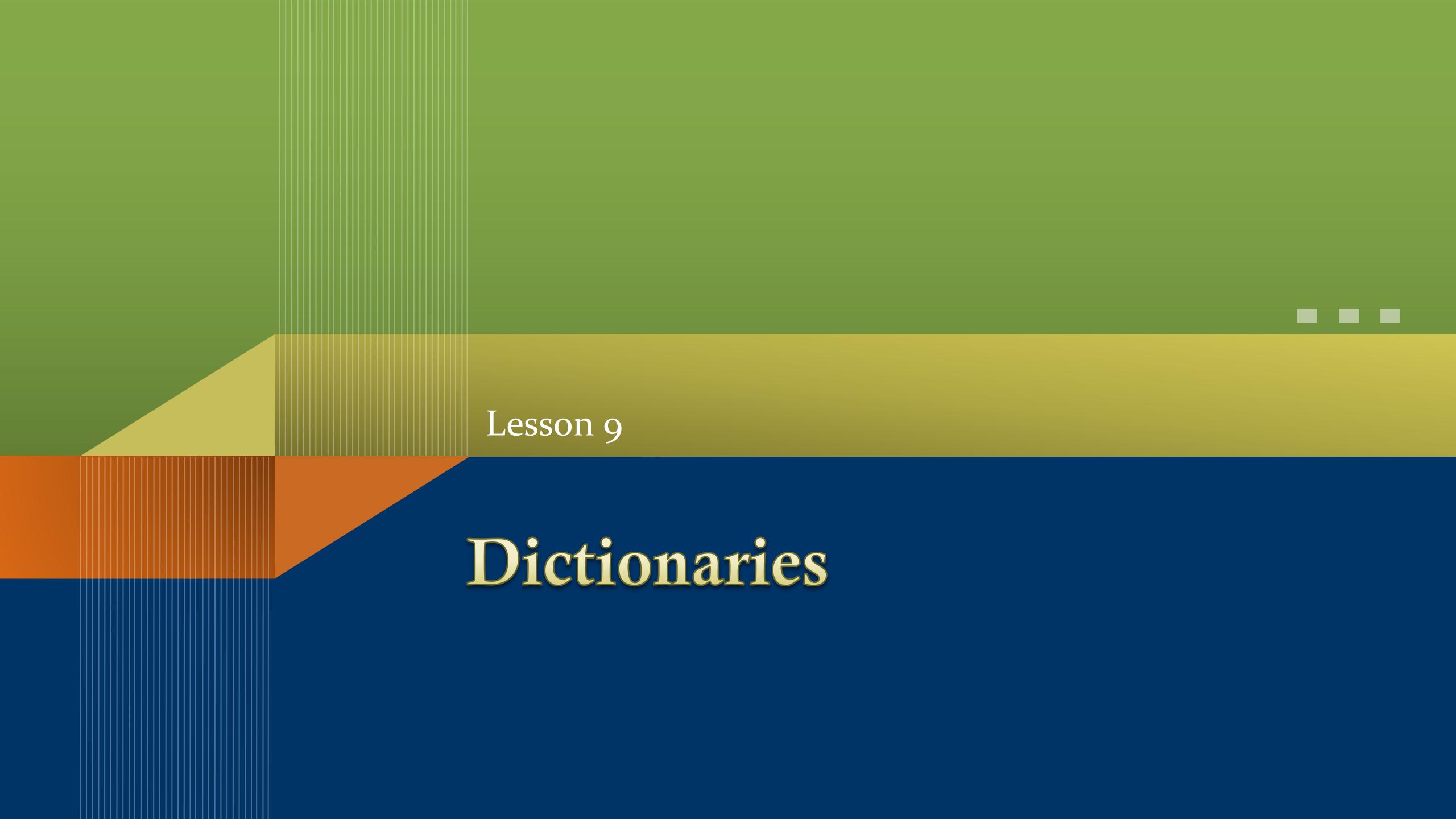
    elif word.endswith(">kr"):
        adr.append(word)

    else:
        continue

    print(adr)
```

Try yourself

- From mails.txt, get all the Months
- From mails.txt, get all the hours
 - 12:30

The background features a decorative graphic on the left side. It consists of a vertical column of thin, light-colored vertical lines. A diagonal line cuts across the graphic, starting from the bottom-left corner and extending upwards towards the top-right. The area below the diagonal is filled with a solid orange color, while the area above it is a lighter yellow. The rest of the slide has a solid green background.

Lesson 9

Dictionaries

Basic Terms

- Keys: variables that are used to call in Values
- Values: variables that get called in by Keys

Lists vs Dictionaries

- Organized according to the order
- Indexes have to be integers
- Order of values work as keys
- [a, b, c]

List

```
>>> myList = ['Python', 'C', 'Java']
>>> myList[0]
'Python'
>>> myList[:2]
['Python', 'C']
>>> |
```

Lists vs Dictionaries

Dictionary

```
>>> countries = {'Asia': 'Korea', 'EU' : 'Germany'}  
>>> countries['Asia']  
'Korea'  
>>> |
```

- Organized according to keys and values
- Indexes can be of any type
- { key1: val1, key2:val2}
- Only one value per key

How to use a Dictionary

- Adding new Key & Value
- Calling values using key
- Modifying the dictionary

How to use a Dictionary

- Adding new Key & Value

Creating and empty dictionary

```
>>> bag = dict()  
>>> print(bag)  
{}  
>>> |
```

Adding new keys and values (similar to append() of lists)

```
>>> bag['money'] = '$'  
>>> bag['key'] = 'value'  
>>> print(bag)  
{'money': '$', 'key': 'value'}  
>>> |
```

How to use a Dictionary

- Calling values using key (indexing)
- In lists, the order of variables work as keys
 - `Name_of_dict['key'] == val`

```
>>> bag['money'] = '$'  
>>> bag['key'] = 'value'  
>>> print(bag)  
{'money': '$', 'key': 'value'}  
>>> |
```

```
>>> print(bag['money'])  
$
```



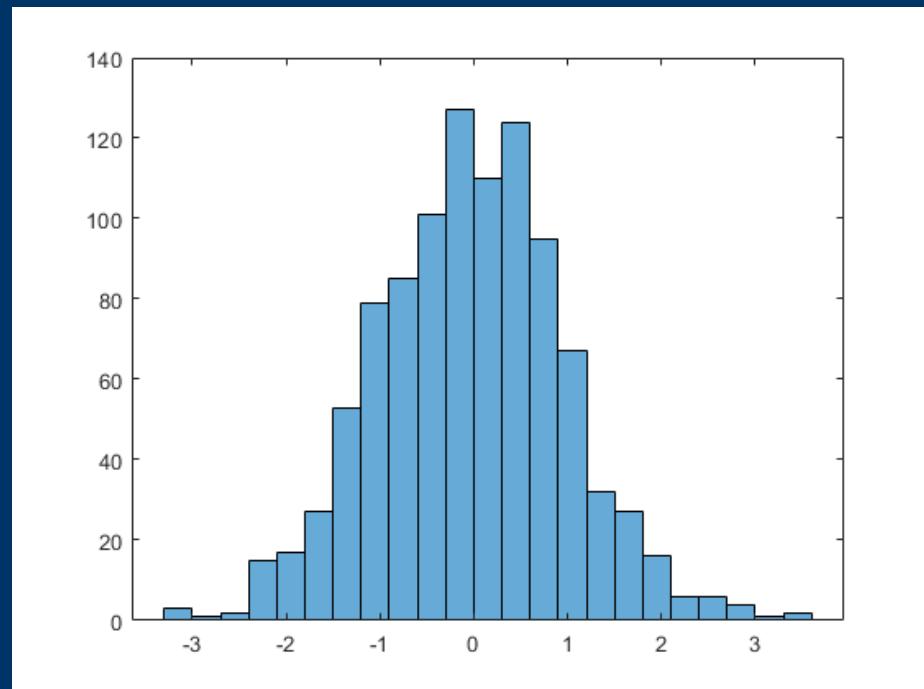
How to use a Dictionary

- Modifying a dictionary
 - We can simply modify our dictionaries by rewriting the keys and values

```
>>> bag = {'money': '$', 'key': 'value'}
>>> bag['money'] = 'won'
>>> print(bag)
{'money': 'won', 'key': 'value'}
>>> |
```

Creating a Histogram

- Histogram : graph used to see the distribution of data
 - Need to count how many times a certain variable appears



Counting Data

- Using dictionaries, it is possible to get the frequency of data
 - If a certain data appears for the first time, we can make a key for that type of data
 - If the data appears again, we can add 1 to the value assigned to the data

cow

dog

chicken

dog

dog

pig

pig

```
count = {}

data = ['cow', 'pig', 'dog', 'dog', 'chicken', 'pig', 'dog']

for animal in data:
    if animal not in count:
        count[animal] = 1
    else:
        count[animal] += 1

print(count)
```

Creating an empty dict

For each variable in data

If that animal is not in the list yet

Make that animal a key with the value of 1

If it is in the list

Add 1 to the value of that key(animal)

The get() method

- The lines of code for counting is used too many times
 - So, people made a method to do it quickly

```
count = {}

data = ['cow', 'pig', 'dog', 'dog', 'chicken', 'pig', 'dog']

for animal in data:
    if animal not in count:
        count[animal] = 1
    else:
        count[animal] += 1

print(count)
```



```
for animal in data:
    count[animal] = count.get(animal, 0) + 1
```

The get() method

for animal in data:

```
count[animal] = count.get(animal, 0) +1
```

```
get(name of key , default value) +1
```

Retrieving lists of Keys and Values

- We can choose if we are going to print keys, values, or both

```
counts = { 'kim': 21, 'fred': 42, 'jan': 100 }
print(list(counts))
print(counts.keys())
print(counts.values())
print(counts.items())
```

```
['kim', 'fred', 'jan']
dict_keys(['kim', 'fred', 'jan'])
dict_values([21, 42, 100])
dict_items([('kim', 21), ('fred', 42), ('jan', 100)])
```



‘for’ loop for dictionaries

- for loop reads keys and values separately

```
counts = { 'kim': 21, 'fred': 42, 'jan': 100}  
  
for key in counts:  
    print(key, counts[key])
```

```
counts = { 'kim': 21, 'fred': 42, 'jan': 100 }  
  
for a,b in counts.items():  
    print(a,b)
```

```
kim 21  
fred 42  
jan 100  
>>> |
```

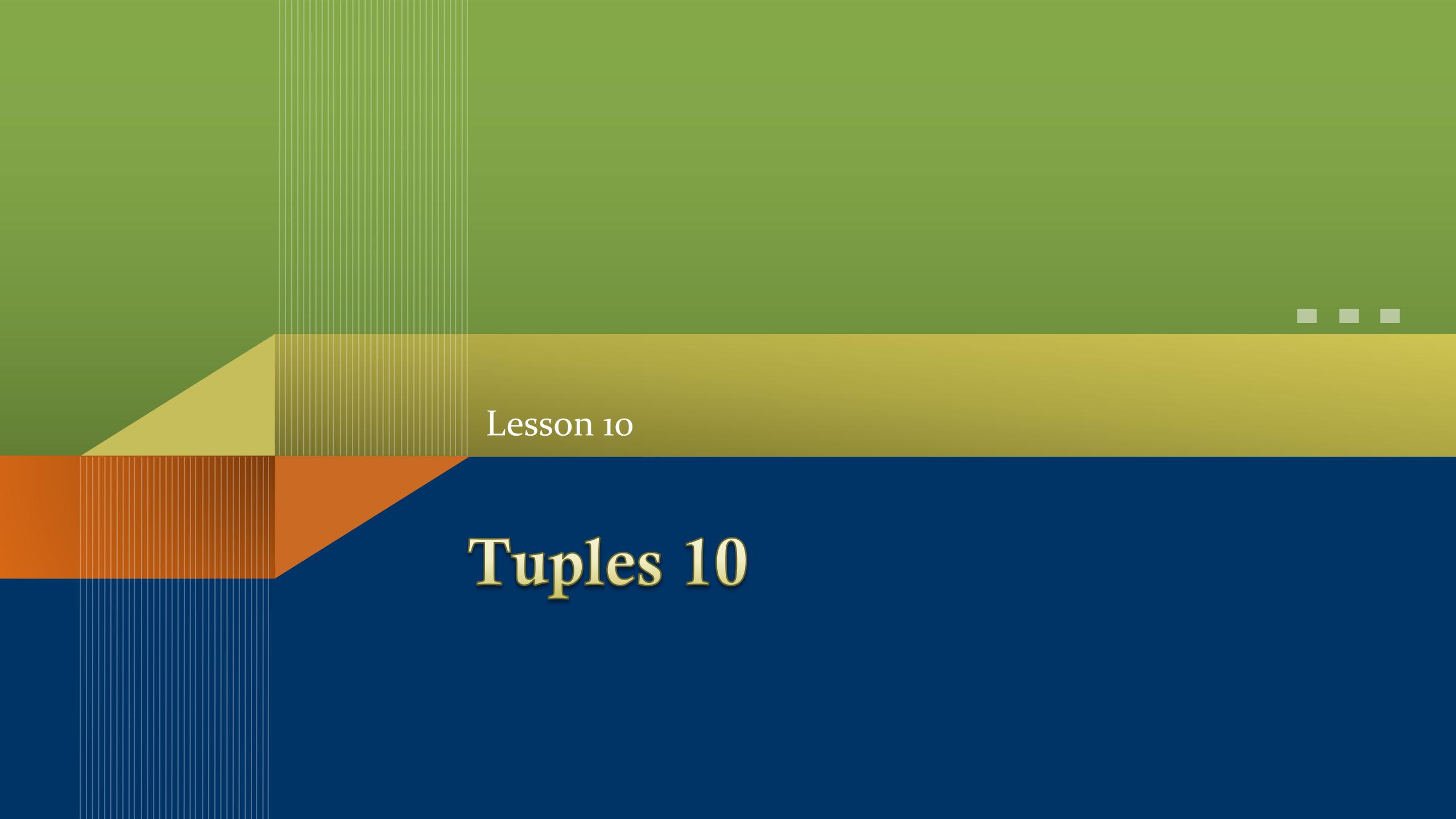
```
name = input('Enter file: ')
handle = open(name)

counts = {}
for line in handle:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word, 0) +1

bigcount = None
bigword = None
for word, count in counts.items():
    if bigcount == None or count > bigcount:
        bigword = word
        bigcount = count
```

Try Yourself

- Write the codes from the previous screen
- Create a textfile, whatever the content is, and try to see which word most frequently appears

The background features a decorative graphic on the left side. It consists of a vertical column of thin, light-colored vertical lines. A diagonal line cuts across the graphic, starting from the bottom-left corner and extending upwards towards the top-right. The area below the diagonal is filled with a solid orange color, while the area above it is a lighter yellow. The rest of the slide has a solid dark blue background.

Lesson 10

Tuples 10

Tuples

- Immutable lists with a round bracket
 - [] vs ()
- Almost all functions work in the same way
 - Except the ones that modify the list

```
x = ['Glenn', 'Sally', 'Joseph']
print(x[2])

y = ('Glenn', 'Sally', 'Joseph')
print(y[2])
```

```
Joseph
Joseph
>>> |
```

```
In [1]: x = [7, 8, 9]
x[2] = 6
print(x)
```

```
[7, 8, 6]
```

```
In [2]: y = (3, 4, 5)
y[2] = 0
print(y)
```

```
-----  
TypeError  
<ipython-input-2-60f2459944ed> in <module>
      1 y = (3, 4, 5)
----> 2 y[2] = 0
      3 print(y)
```

```
Traceback (most recent call last)
```

```
TypeError: 'tuple' object does not support item assignment
```

Why do we use Tuples?

- They are more efficient than lists (for processing)
- They take up less memory and perform better
- Often used when storing temporary data

When tuples come to the left

- We can assign multiple values to multiple variables when using tuples

```
In [3]: (x,y) = (4, 'cake')
print(y)
```

cake

```
In [8]: d = {}
d['alpha'] = 1
d['beta'] = 6
for (k,v) in d.items():
    print(k,v)
```

alpha 1
beta 6

```
tuples = d.items()
print(tuples)
```

```
dict_items([('alpha', 1), ('beta', 6)])
```

Comparing Tuples

```
In [10]: (0, 1, 2) < (5, 1, 2)
```

```
Out [10]: True
```

```
In [11]: (0, 1, 150349) < (0, 2, -19)
```

```
Out [11]: True
```

```
In [13]: ("Sam", "Jay") < ('Sally', 'Jim')
```

```
Out [13]: False
```

Sorting List of Tuples

- List of tuples: we've seen it many times in dictionary
- We can sort tuples to come in an alphabetical/numerical order
 - sorted() only sorts by keys

```
In [20]: d = {'a':15, 'b': 21, 'c': 12, 'ab': 13}
d.items()
Out [20]: dict_items([('a', 15), ('b', 21), ('c', 12), ('ab', 13)])
```

```
In [21]: d = {'a':15, 'b': 21, 'c': 12, 'ab': 13}
d.items()
sorted(d.items())
Out [21]: [('a', 15), ('ab', 13), ('b', 21), ('c', 12)]
```

- How can we sort by Values?
 - Flip it

```
In [22]: file = open("    .txt")
counts = {}
for line in file:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word, 0) + 1

list = []
for k,v in counts.items():
    newtuple = (v,k)
    list.append(newtuple)
list = sorted(list, reverse = True)

for v,k in list[:10]:
    print(k,v)
```

Sorting in a descending order



DATUM

Yonsei Data Science Academia

NUMPY

DATUM

IMPORT STATEMENTS?

Why is this “IMPORT” statement important?

- It helps you get access to the advance python “tools” - modules

You can easily `import` any of them by using this syntax:

```
import [module_name]
```

eg. `import random`

QUICK REVIEW

```
x = 3
print(type(x)) # Prints <class 'int'>
print(x) # Prints 3
print(x + 1) # Addition; prints 4
print(x - 1) # Subtraction; prints 2
print(x * 2) # Multiplication; prints 6
print(x ** 2) # Exponentiation; prints 9
x += 1
print(x) # Prints 4
x *= 2
print(x) # Prints 8
y = 2.5
print(type(y)) # Prints <class 'float'>
print(y, y + 1, y * 2, y ** 2) # Prints 2.5 3.5 5.0 6.25
```

QUICK REVIEW

```
xs = [3, 1, 2]      # Create a list
print(xs, xs[2])  # Prints "[3, 1, 2] 2"
print(xs[-1])     # Negative indices count from the end of the list;
xs[2] = 'foo'       # Lists can contain elements of different types
print(xs)          # Prints "[3, 1, 'foo']"
xs.append('bar')   # Add a new element to the end of the list
print(xs)          # Prints "[3, 1, 'foo', 'bar']"
x = xs.pop()       # Remove and return the last element of the list
print(x, xs)       # Prints "bar [3, 1, 'foo']"
```

QUICK REVIEW

```
nums = list(range(5))          # range is a built-in function that creates a list of integers
print(nums)                    # Prints "[0, 1, 2, 3, 4]"
print(nums[2:4])               # Get a slice from index 2 to 4 (exclusive); prints "[2, 3]"
print(nums[2:])                # Get a slice from index 2 to the end; prints "[2, 3, 4]"
print(nums[:2])                # Get a slice from the start to index 2 (exclusive); prints "[0, 1]"
print(nums[:])                 # Get a slice of the whole list; prints "[0, 1, 2, 3, 4]"
print(nums[:-1])               # Slice indices can be negative; prints "[0, 1, 2, 3]"
nums[2:4] = [8, 9]              # Assign a new sublist to a slice
print(nums)                    # Prints "[0, 1, 8, 9, 4]"
```

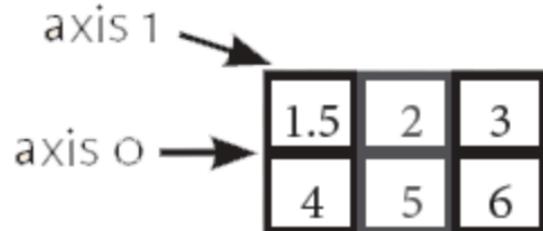
WHAT IS NUMPY

Library to perform mathematical and statistical operations. Works for multi-dimensional arrays. Supports mathematical operations & arrays & matrices

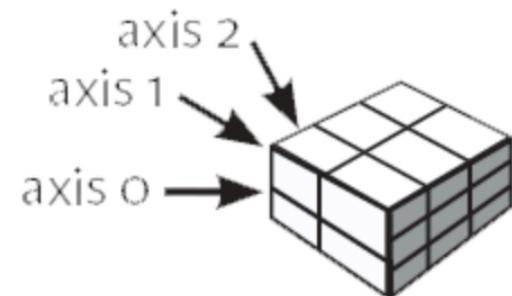
1D array



2D array



3D array



HOW TO INSTALL NUMPY

FOR MAC

/usr/bin/ruby -e "\$(curl -fsSL

<https://raw.githubusercontent.com/Homebrew/install/master/install>)"

```
● ● ● sohnshine — ruby -e #!/usr/bin/ruby\012# This script installs to /usr/local only. To...
Last login: Wed Jul 24 00:08:01 on ttys000
Eurys-MacBook:~ sohnshine$ /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
==> This script will install:
/usr/local/bin/brew
/usr/local/share/doc/homebrew
/usr/local/share/man/man1/brew.1
/usr/local/share/zsh/site-functions/_brew
/usr/local/etc/bash_completion.d/brew
/usr/local/Homebrew

Press RETURN to continue or any other key to abort
```

HOW TO FOR WINDOWS INSTALL NUMPY

<https://solarianprogrammer.com/2017/02/25/install-numpy-scipy-matplotlib-python-3-windows/>

```
1  python -m pip install numpy
2  python -m pip install scipy
3  python -m pip install matplotlib
```

THE BASICS

Main object is the homogenous multidimensional array.

These dimensions are called axes.

[1, 2, 1]

The coordinates of a point in 3D spaces has one axis. That axis has 3 elements in it, so we say it has a length of 3.

ARRAY

NjmPy's array class is called `ndarray`- which is also known as just "array". Here are some important attributes of `ndarray`.

`ndarray.ndim`

the number of axes (dimensions) of the array.

`ndarray.shape`

the dimensions of the array. This is a tuple of integers indicating the size of the array in each dimension. For a matrix with n rows and m columns, `shape` will be `(n, m)`. The length of the `shape` tuple is therefore the number of axes, `ndim`.

`ndarray.size`

the total number of elements of the array. This is equal to the product of the elements of `shape`.

EXAMPLE

EXPLANATION

```
>>> import numpy as np
>>> a = np.arange(15).reshape(3, 5)
>>> a
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
>>> a.shape
(3, 5)
>>> a.ndim
2
```

Import the module

Set a array range from 0-15 with axes 3 & 5

Ask python about its size

Ask python about dimension

EXAMPLE

EXPLANATION

```
>>> type(a)  
<type 'numpy.ndarray'>  
>>> b = np.array([6, 7, 8])  
>>> b  
array([6, 7, 8])  
>>> type(b)  
<type 'numpy.ndarray'>
```

Ask python what type "a" is
(previously, we set the variable a as
the array
Set array as "b"

Check type

WAY TO CREATE ARRAY

```
>>> import numpy as np
>>> a = np.array([2,3,4])
>>> a
array([2, 3, 4])
>>> a.dtype
dtype('int64')
>>> b = np.array([1.2, 3.5, 5.1])
>>> b.dtype
dtype('float64')
```

```
>>> a = np.array(1,2,3,4)      # WRONG
>>> a = np.array([1,2,3,4])   # RIGHT
```

WAY TO CREATE ARRAY

`array` transforms sequences of sequences into two-dimensional arrays, sequences of sequences of sequences into three-dimensional arrays, and so on.

```
>>> b = np.array([(1.5,2,3), (4,5,6)])
>>> b
array([[ 1.5,  2. ,  3. ],
       [ 4. ,  5. ,  6. ]])
```

WAY TO CREATE ARRAY

Numpy offers functions to create initial placeholder content

Function zero creates array full of zeros

Function ones create an array full of ones

And function empty creates an array whose initial content is
random depending on the state of memory.

```
>>> np.zeros( (3,4) )                                     >>>
array([[ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.]])
>>> np.ones( (2,3,4), dtype=np.int16 )                  # dtype can also
be specified
array([[[ 1,  1,  1,  1],
       [ 1,  1,  1,  1],
       [ 1,  1,  1,  1]],
      [[ 1,  1,  1,  1],
       [ 1,  1,  1,  1],
       [ 1,  1,  1,  1]]], dtype=int16)
>>> np.empty( (2,3) )                                  # uninitialized,
output may vary
array([[ 3.73603959e-262,   6.02658058e-154,   6.55490914e-260],
       [ 5.30498948e-313,   3.14673309e-307,   1.00000000e+000]])
```

WAY TO CREATE ARRAY

Like before, we can also use range

```
>>> np.arange( 10, 30, 5 )  
array([10, 15, 20, 25])  
>>> np.arange( 0, 2, 0.3 )          # it accepts float argument  
s  
array([ 0. ,  0.3,  0.6,  0.9,  1.2,  1.5,  1.8])
```

PROBLEM WITH ARANGE

When we use `arrange`, it is hard to guess the number of elements obtained, do so we'll use `linspace`. This receives as an argument the number of elements that we want

```
>>> from numpy import pi
>>> np.linspace( 0, 2, 9 )                      # 9 numbers from 0 to 2
array([ 0. ,  0.25,  0.5 ,  0.75,  1. ,  1.25,  1.5 ,  1.75,  2. ])
>>> x = np.linspace( 0, 2*pi, 100 )            # useful to evaluate function at lots of points
>>> f = np.sin(x)
```

RESHAPE

It always follows the following layout, last axis is printed from left to right
Second to last is printed from top to bottom

```
>>> a = np.arange(6)                      # 1d array
>>> print(a)
[0 1 2 3 4 5]
>>>
>>> b = np.arange(12).reshape(4,3)        # 2d array
>>> print(b)
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
```

BASIC OPERATIONS

Arithmetic operators on arrays apply elementwise

```
>>> a = np.array( [20,30,40,50] )
>>> b = np.arange( 4 )
>>> b
array([0, 1, 2, 3])
>>> c = a-b
>>> c
array([20, 29, 38, 47])
>>> b**2
array([0, 1, 4, 9])
>>> 10*np.sin(a)
array([ 9.12945251, -9.88031624,  7.4511316 , -2.62374854])
>>> a<35
array([ True, True, False, False])
```

BASIC OPERATIONS

```
>>> a = np.random.random(2,3)  
>>> a  
array([[ 0.18626021,  0.34556073,  0.39676747],  
       [ 0.53881673,  0.41919451,  0.6852195 ]])  
>>> a.sum()  
2.5718191614547998  
>>> a.min()  
0.1862602113776709  
>>> a.max()  
0.6852195003967595
```

BASIC OPERATIONS

Slicing and iteration is same as lists

```
>>> a = np.arange(10)**3
>>> a
array([  0,    1,    8,   27,   64,  125,  216,  343,  512,  729])
>>> a[2]
8
>>> a[2:5]
array([ 8, 27, 64])
```

STACKING AN ARRAY

```
>>> a = np.floor(10*np.random.random((2,2)))
>>> a
array([[ 8.,  8.],
       [ 0.,  0.]])
>>> b = np.floor(10*np.random.random((2,2)))
>>> b
array([[ 1.,  8.],
       [ 0.,  4.]])
>>> np.vstack((a,b))
array([[ 8.,  8.],
       [ 0.,  0.],
       [ 1.,  8.],
       [ 0.,  4.]])
>>> np.hstack((a,b))
array([[ 8.,  8.,  1.,  8.],
       [ 0.,  0.,  0.,  4.]])
```

```
>>> from numpy import newaxis
>>> np.column_stack((a,b))      # with 2D arrays
array([[ 8.,  8.,  1.,  8.],
       [ 0.,  0.,  0.,  4.]])
>>> a = np.array([4.,2.])
>>> b = np.array([3.,8.])
>>> np.column_stack((a,b))      # returns a 2D array
array([[ 4.,  3.],
       [ 2.,  8.]])
>>> np.hstack((a,b))          # the result is different
array([ 4.,  2.,  3.,  8.])
>>> a[:,newaxis]               # this allows to have a 2D columns vect
or
array([[ 4.],
       [ 2.]])
>>> np.column_stack((a[:,newaxis],b[:,newaxis]))
array([[ 4.,  3.],
       [ 2.,  8.]])
>>> np.hstack((a[:,newaxis],b[:,newaxis]))  # the result is the same
array([[ 4.,  3.],
       [ 2.,  8.]])
```

```
>>> a = np.floor(10*np.random.random((2,12)))
>>> a
array([[ 9.,  5.,  6.,  3.,  6.,  8.,  0.,  7.,  9.,  7.,  2.,  7.],
       [ 1.,  4.,  9.,  2.,  2.,  1.,  0.,  6.,  2.,  2.,  4.,  0.]])
>>> np.hsplit(a,3)  # Split a into 3
[array([[ 9.,  5.,  6.,  3.],
       [ 1.,  4.,  9.,  2.]]), array([[ 6.,  8.,  0.,  7.],
       [ 2.,  1.,  0.,  6.]]), array([[ 9.,  7.,  2.,  7.],
       [ 2.,  2.,  4.,  0.]])]
>>> np.hsplit(a,(3,4))  # Split a after the third and the fourth column
[array([[ 9.,  5.,  6.],
       [ 1.,  4.,  9.]]), array([[ 3.],
       [ 2.]]), array([[ 6.,  8.,  0.,  7.,  9.,  7.,  2.,  7.],
       [ 2.,  1.,  0.,  6.,  2.,  2.,  4.,  0.]])]
```

BOOLEAN ARRAYS

```
>>> a = np.arange(10)**3
>>> a
array([  0,    1,    8,   27,   64,  125,  216,  343,  512,  729])
>>> a[2]
8
>>> a[2:5]
array([ 8, 27, 64])
```

BASIC OPERATIONS

Slicing and iteration is same as lists

```
>>> a = np.arange(10)**3
>>> a
array([  0,    1,    8,   27,   64,  125,  216,  343,  512,  729])
>>> a[2]
8
>>> a[2:5]
array([ 8, 27, 64])
```

CHALLENGE

1. Create an array of 10 zeros
2. Create an array of 10 ones
3. Create an array of integers from 10 to 50
4. Create a 3x3 matrix with values ranging from 0 to 8
5. Create an array of 20 linearly spaced points between 0 and 1

Import NumPy as np

```
: import numpy as np
```

Create an array of 10 zeros

```
: np.zeros(10)
```

```
: array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

Create an array of 10 ones

```
: np.ones(10)
```

```
: array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```



Create an array of the integers from 10 to 50

In [7]: `np.arange(10,51)`

Out[7]: `array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50])`

Create an array of all the even integers from 10 to 50

In [8]: `np.arange(10,51,2)`

Out[8]: `array([10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50])`



Create an array of 20 linearly spaced points between 0 and 1:

In [16]: `np.linspace(0,1,20)`

Out[16]: `array([0.0, 0.05263158, 0.10526316, 0.15789474, 0.21052632, 0.26315789, 0.31578947, 0.36842105, 0.42105263, 0.47368421, 0.52631579, 0.57894737, 0.63157895, 0.68421053, 0.73684211, 0.78947368, 0.84210526, 0.89473684, 0.94736842, 1.0])`

CHALLENGE PART 2

Create an Array that looks like this:

```
array([[ 1,  2,  3,  4,  5],  
       [ 6,  7,  8,  9, 10],  
       [11, 12, 13, 14, 15],  
       [16, 17, 18, 19, 20],  
       [21, 22, 23, 24, 25]])
```



DATUM

Yonsei Data Science Academia

PANDAS

DATUM

PANDAS DATA STRUCTURES

TWO DIFFERENT TYPES OF DATA STRUCTURE IN PANDAS: SERIES AND DATA FRAMES

Series: a one dimensional data structure (one dimensional array)

Data Frame: two or more dimensions. (table with rows & columns)

In [4]: test_set_series

Out[4]:

0	15
1	36
2	41
3	14
4	69
5	73
6	92
7	56
8	101

In [12]: big_table

Out[12]:

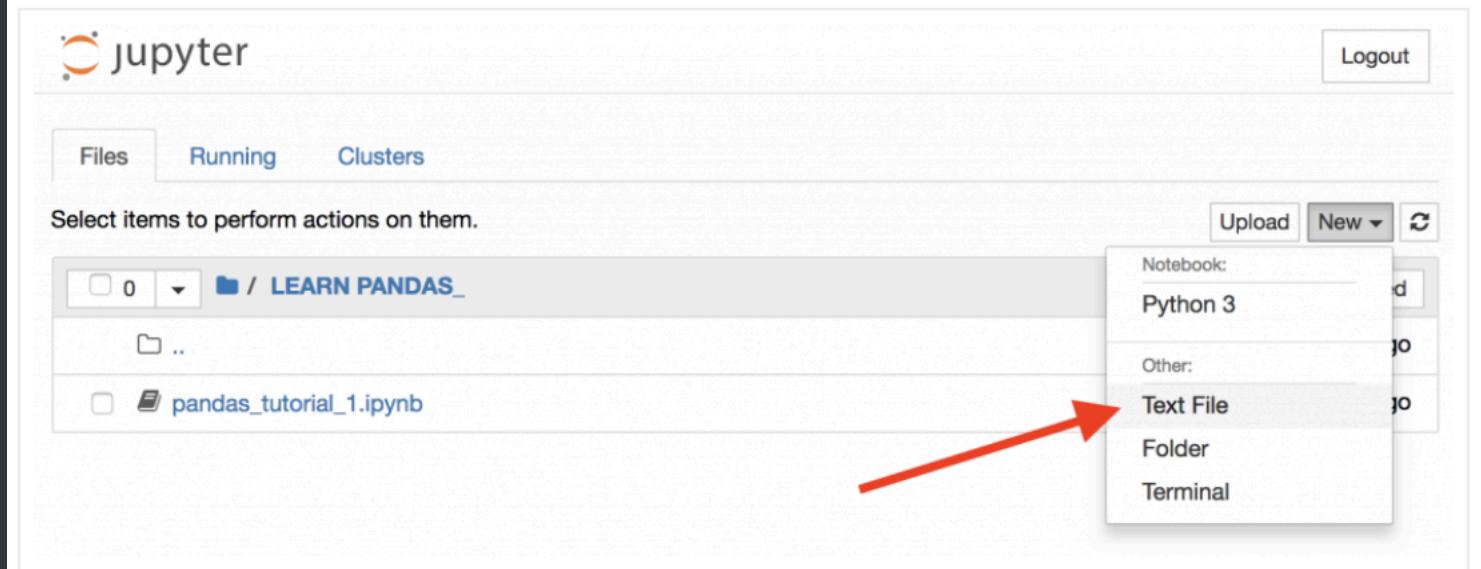
	user_id	phone_type	source	free	super
0	1000001	android	invite_a_friend	5.0	0.0
1	1000002	ios	invite_a_friend	4.0	0.0
2	1000003	error	invite_a_friend	37.0	0.0

CREATE YOUR OWN CSV FILE

```
animal,uniq_id,water_nee  
d elephant,1001,500  
elephant,1002,600  
elephant,1003,550  
tiger,1004,300  
tiger,1005,320  
tiger,1006,330  
tiger,1007,290  
tiger,1008,310  
zebra,1009,200  
zebra,1010,220
```

CREATE YOUR OWN CSV FILE

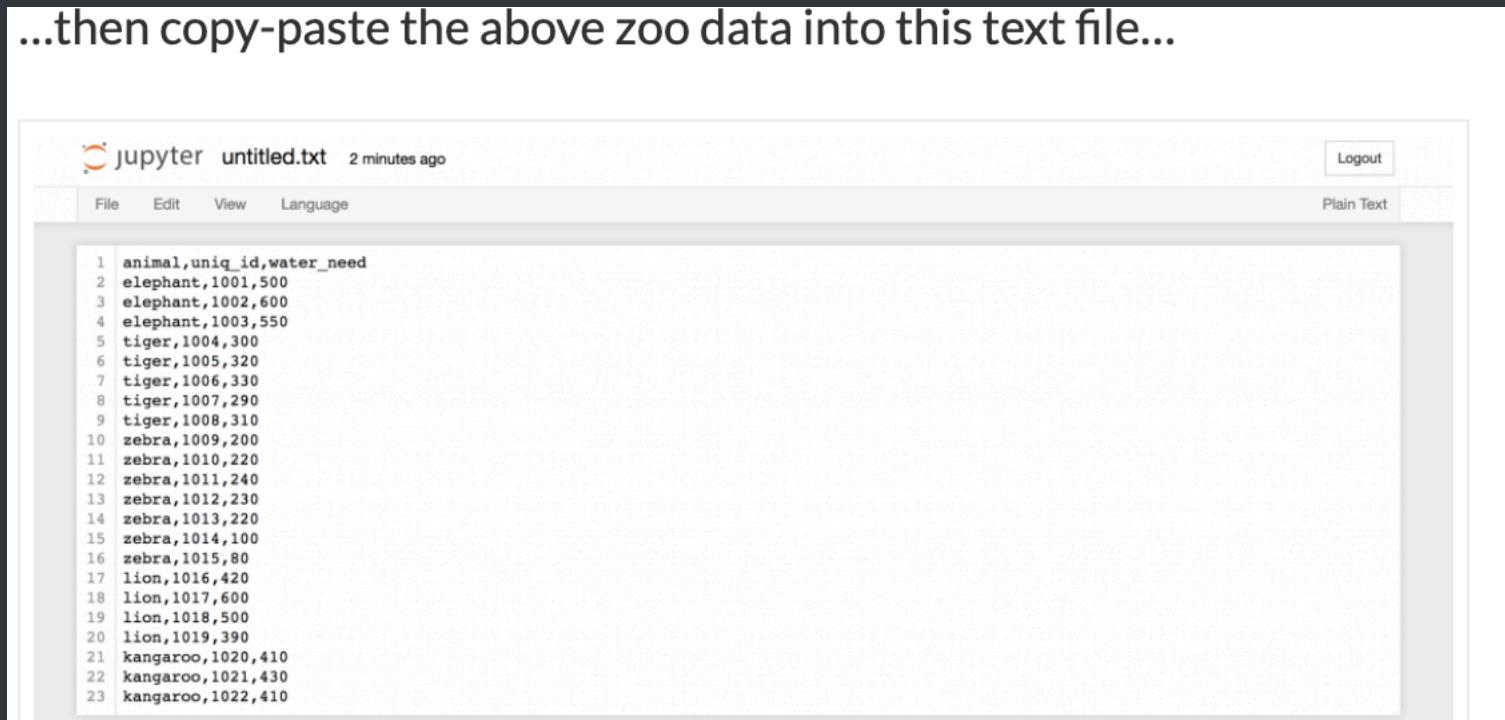
Go back to your Jupyter Home tab and create a new text file...



The image shows the Jupyter Home interface. At the top, there is a navigation bar with the Jupyter logo, a 'Logout' button, and tabs for 'Files', 'Running', and 'Clusters'. Below the navigation bar, a message says 'Select items to perform actions on them.' On the left, there is a file list showing a folder named 'LEARN PANDAS_' containing an 'ipynb' file named 'pandas_tutorial_1.ipynb'. On the right, a context menu is open with the following options: 'Upload', 'New', 'Notebook: Python 3', 'Other: Text File', 'Other: Folder', and 'Other: Terminal'. A red arrow points to the 'Text File' option in the 'Other' section of the menu.

CREATE YOUR OWN CSV FILE

...then copy-paste the above zoo data into this text file...

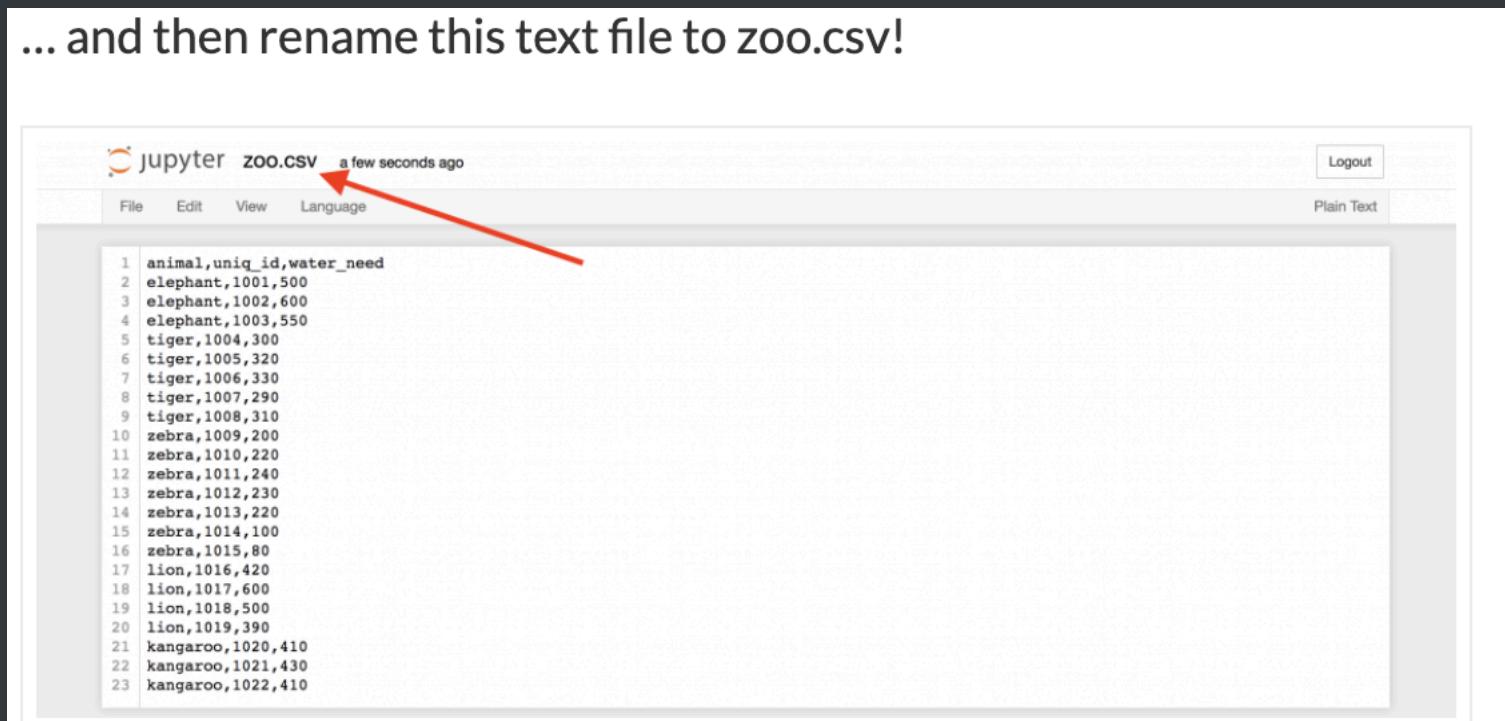


The screenshot shows a Jupyter Notebook interface with a single text cell. The cell title is 'untitled.txt' and it was created '2 minutes ago'. The cell content is a CSV file with 23 rows of data, each containing an animal name, a unique ID, and a water need value. The data is as follows:

	animal	uniq_id	water_need
1	elephant	1001	500
2	elephant	1002	600
3	elephant	1003	550
4	tiger	1004	300
5	tiger	1005	320
6	tiger	1006	330
7	tiger	1007	290
8	tiger	1008	310
9	zebra	1009	200
10	zebra	1010	220
11	zebra	1011	240
12	zebra	1012	230
13	zebra	1013	220
14	zebra	1014	100
15	zebra	1015	80
16	lion	1016	420
17	lion	1017	600
18	lion	1018	500
19	lion	1019	390
20	kangaroo	1020	410
21	kangaroo	1021	430
22	kangaroo	1022	410

CREATE YOUR OWN CSV FILE

... and then rename this text file to `zoo.csv`!



The screenshot shows a Jupyter Notebook interface. At the top, there is a header with the text "jupyter" and "zoo.csv" followed by "a few seconds ago". A red arrow points from the text "zoo.csv" in the header to the file name "zoo.csv" in the code block below. The code block contains the following text:

```
1 animal,uniq_id,water_need
2 elephant,1001,500
3 elephant,1002,600
4 elephant,1003,550
5 tiger,1004,300
6 tiger,1005,320
7 tiger,1006,330
8 tiger,1007,290
9 tiger,1008,310
10 zebra,1009,200
11 zebra,1010,220
12 zebra,1011,240
13 zebra,1012,230
14 zebra,1013,220
15 zebra,1014,100
16 zebra,1015,80
17 lion,1016,420
18 lion,1017,600
19 lion,1018,500
20 lion,1019,390
21 kangaroo,1020,410
22 kangaroo,1021,430
23 kangaroo,1022,410
```

READING THE FILES

READING THE CSV FILE

Again, the function that you have to use is: `read_csv()`

Type this to a new cell:

```
pd.read_csv('zoo.csv', delimiter = ',')
```

```
In [2]: import numpy as np  
import pandas as pd
```

```
In [14]: pd.read_csv('zoo.csv', delimiter=',')
```

```
Out[14]:
```

	animal	uniq_id	water_need
0	elephant	1001	500
1	elephant	1002	600
2	elephant	1003	550
3	tiger	1004	300
4	tiger	1005	320
5	tiger	1006	330
6	tiger	1007	290
7	tiger	1008	310
8	zebra	1009	200
9	zebra	1010	220
10	zebra	1011	240
11	zebra	1012	230
12	zebra	1013	220
13	zebra	1014	100
14	zebra	1015	80

SETTING OUR OWN HEADINGS

Most of time, we will be using a table that someone else has already created. How do we do this?

Download the file in our kakotalk chatroom (one titled
[pandas_tutorial_read.csv])

Drag onto your jupyter notebook to same in side of it.

IF IT SAYS IT DOESN'T HAVE
PANDZS, USE

sudo pip3 install pandas

OPENING THE FILE BY USING READ_CSV

```
pd.read_csv('pandas_tutorial_read.csv', delimiter=';')
```

The data is loaded into pandas!

```
In [3]: pd.read_csv('pandas_tutorial_read.csv', delimiter=';')
```

```
Out[3]:
```

	2018-01-01 00:01:01	read	country_7	2458151261	SEO	North America
0	2018-01-01 00:03:20	read	country_7	2458151262	SEO	South America
1	2018-01-01 00:04:01	read	country_7	2458151263	AdWords	Africa
2	2018-01-01 00:04:02	read	country_7	2458151264	AdWords	Europe
3	2018-01-01 00:05:03	read	country_8	2458151265	Reddit	North America
4	2018-01-01 00:05:42	read	country_6	2458151266	Reddit	North America
5	2018-01-01 00:06:06	read	country_2	2458151267	Reddit	Europe
6	2018-01-01 00:06:15	read	country_6	2458151268	AdWords	Europe
7	2018-01-01 00:07:21	read	country_7	2458151269	AdWords	North America
8	2018-01-01 00:07:29	read	country_5	2458151270	Reddit	North America
9	2018-01-01 00:07:57	read	country_5	2458151271	AdWords	Asia
10	2018-01-01 00:08:57	read	country_7	2458151272	SEO	Australia

No headers!

OPENING THE FILE BY USING READ_CSV

```
pd.read_csv('pandas_tutorial_read.csv', delimiter=';',
            names = ['my_datetime', 'event', 'country', 'user_id',
            'source', 'topic'])
```

```
In [21]: pd.read_csv('pandas_tutorial_read.csv', delimiter=';',
                     names = ['my_datetime', 'event', 'country', 'user_id', 'source', 'topic'])
```

```
Out[21]:
```

my_datetime	event	country	user_id	source	topic
-------------	-------	---------	---------	--------	-------

I CAN ALWAYS READ THE ARTICLE

In [27]: article_read

Out[27]:

	my_datetime	event	country	user_id	source	topic
0	2018-01-01 00:01:01	read	country_7	2458151261	SEO	North America
1	2018-01-01 00:03:20	read	country_7	2458151262	SEO	South America
2	2018-01-01 00:04:01	read	country_7	2458151263	AdWords	Africa

FEW FUNCTIONS YOU CAN DO

the first 5 lines – by typing:

```
article_read.head()
```

In [28]: `article_read.head()`

Out[28]:

	my_datetime	event	country	user_id	source	topic
0	2018-01-01 00:01:01	read	country_7	2458151261	SEO	North America
1	2018-01-01 00:03:20	read	country_7	2458151262	SEO	South America
2	2018-01-01 00:04:01	read	country_7	2458151263	AdWords	Africa
3	2018-01-01 00:04:02	read	country_7	2458151264	AdWords	Europe
4	2018-01-01 00:05:03	read	country_8	2458151265	Reddit	North America

FEW FUNCTIONS YOU CAN DO

article_read.tail()

In [29]: article_read.tail()

Out[29]:

	my_datetime	event	country	user_id	source	topic
1790	2018-01-01 23:57:14	read	country_2	2458153051	AdWords	North America
1791	2018-01-01 23:58:33	read	country_8	2458153052	SEO	Asia
1792	2018-01-01 23:59:36	read	country_6	2458153053	Reddit	Asia
1793	2018-01-01 23:59:36	read	country_7	2458153054	AdWords	Europe
1794	2018-01-01 23:59:38	read	country_5	2458153055	Reddit	Asia

FEW FUNCTIONS YOU CAN DO

```
article_read.sample(5)
```

```
In [32]: article_read.sample(5)
```

```
Out[32]:
```

	my_datetime	event	country	user_id	source	topic
277	2018-01-01 03:43:16	read	country_5	2458151538	Reddit	Africa
996	2018-01-01 13:26:57	read	country_7	2458152257	Reddit	Africa
373	2018-01-01 05:03:06	read	country_6	2458151634	SEO	North America
475	2018-01-01 06:24:08	read	country_2	2458151736	Reddit	Europe
847	2018-01-01 11:28:21	read	country_4	2458152108	Reddit	Asia



ONLY PRINTING SPECIFIC COLUMNS

```
article_read[['country', 'user_id']]
```

```
In [56]: article_read[['country', 'user_id']]
```

```
Out[56]:
```

	country	user_id
0	country_7	2458151261
1	country_7	2458151262
2	country_7	2458151263
3	country_7	2458151264
4	country_8	2458151265
5	country_6	2458151266

FILTER RESULTS

Let's say, you want to see a list of only the users who came from the 'SEO' source. In this case you have to filter for the 'SEO' value in the 'source' column:

```
article_read[article_read.source == 'SEO']
```

FILTER RESULTS

```
In [69]: article_read.source == 'SEO'
```

```
Out[69]: 0      True
         1      True
         2     False
         3     False
         4     False
         5     False
         6     False
         7     False
         8     False
         9     False
        10    False
```

FILTER RESULTS

STEP 2) Then from the `article_read` table, it prints every row where this value is `True` and doesn't print any row where it's `False`.

```
In [70]: article_read[article_read.source == 'SEO']
```

Out[70]:

	my_datetime	event	country	user_id	source	topic
0	2018-01-01 00:01:01	read	country_7	2458151261	SEO	North America
1	2018-01-01 00:03:20	read	country_7	2458151262	SEO	South America
11	2018-01-01 00:08:57	read	country_7	2458151272	SEO	Australia
15	2018-01-01 00:11:22	read	country_7	2458151276	SEO	North America
16	2018-01-01 00:13:05	read	country_8	2458151277	SEO	North America
18	2018-01-01 00:13:39	read	country_4	2458151279	SEO	North America
26	2018-01-01 00:20:18	read	country_5	2458151287	SEO	North America

CHALLENGE

Select the `user_id`, the `country` and the `topic` columns for the users who are from `country_2`! Print the first five rows only!

SOLUTION

```
article_read[article_read.country == 'country_2']  
[['user_id', 'topic', 'country']].head()
```

```
ar_filtered = article_read[article_read.country ==  
'country_2']  
ar_filtered_cols = ar_filtered[['user_id', 'topic',  
'country']]  
ar_filtered_cols.head()
```

SO

One-liner solution

```
In [77]: article_read[article_read.country == 'country_2'][['user_id', 'topic', 'country']].head()
```

```
Out[77]:
```

	user_id	topic	country
6	2458151267	Europe	country_2
13	2458151274	Europe	country_2
17	2458151278	Asia	country_2
19	2458151280	Asia	country_2
20	2458151281	Asia	country_2

More transparent solution (alternative solution)

```
In [78]: ar_filtered = article_read[article_read.country == 'country_2']
```

```
In [79]: ar_filtered_cols = ar_filtered[['user_id', 'topic', 'country']]
```

```
In [80]: ar_filtered_cols.head()
```

```
Out[80]:
```

	user_id	topic	country
6	2458151267	Europe	country_2
13	2458151274	Europe	country_2
17	2458151278	Asia	country_2
19	2458151280	Asia	country_2
20	2458151281	Asia	country_2

JAFAR DE



SOLUTION

Either way, the logic is the same.

First you take your original dataframe (article_read.

Then you filter for the rows where the country value is country_2 ([article_read.country == 'country_2'])

Then you take the three columns that were required ([['user_id', 'topic', 'country']]) and eventually you take the first five rows only (.head()).

FUNCTIONS CAN BE USED AFTER EACH OTHER

```
article_read.head() [['country', 'user_id']]
```

This line first selects the first 5 rows of our data set. And then it takes only the 'country' and the 'user_id' columns.

LET'S NOW GO BACK TO
OUR ZOO DATA SET!

LOADING OUR ZOO DATASET

```
pd.read_csv('zoo.csv', delimiter = ',')
```

```
In [1]: import numpy as np  
import pandas as pd
```

```
In [4]: pd.read_csv('zoo.csv', delimiter = ',')
```

```
Out[4]:      animal  uniq_id  water_need
```

	animal	uniq_id	water_need
0	elephant	1001	500

```
In [1]: import numpy as np  
import pandas as pd
```

```
In [5]: zoo = pd.read_csv('zoo.csv', delimiter = ',')
```

FIVE THINGS WE AIM TO DO

1. Let's count the number of rows (the number of animals) in `zoo` !
2. Let's calculate the total `water_need` of the animals!
3. Let's find out which is the smallest `water_need` value!
4. And then the greatest `water_need` value!
5. And eventually the average `water_need` !

.COUNT()

```
zoo.count()
```

```
In [14]: zoo.count()  
Out[14]: animal      22  
          uniq_id     22  
          water_need  22  
          dtype: int64
```

,count() function counts the number of values in each column

.COUNT() SPECIFY

```
zoo[['animal']].count()
```

```
In [17]: zoo[['animal']].count()
```

```
Out[17]: animal    22
          dtype: int64
```

Making your output clearer by selecting an exact coloum

.SUM()

Following the same logic, you can easily sum the values in the `water_need` column by typing:

```
zoo.water_need.sum()
```

```
In [19]: zoo.water_need.sum()
```

```
Out[19]: 7650
```

.MIN() .MAX()

What's the smallest value in the `water_need` column? I bet you have figured it out already:

```
zoo.water_need.min()
```

```
In [21]: zoo.water_need.min()
```

```
Out[21]: 80
```

And the max value is pretty similar:

```
zoo.water_need.max()
```

```
In [22]: zoo.water_need.max()
```

```
Out[22]: 600
```

.MEAN() .MEDIAN()

```
zoo.water_need.mean()
```

```
In [25]: zoo.water_need.mean()
```

```
Out[25]: 347.72727272727275
```

```
zoo.water_need.median()
```

```
In [26]: zoo.water_need.median()
```

```
Out[26]: 325.0
```

.GROUPBY

GROUPBY KEYWORD

Segregating into groups and finding the mean

```
In [32]: zoo.groupby('animal').mean()
```

```
Out[32]:
```

animal	uniq_id	water_need
elephant	1002.0	550.000000
kangaroo	1021.0	416.666667
lion	1017.5	477.500000
tiger	1006.0	310.000000
zebra	1012.0	184.285714

MERGING

CREATE YOUR OWN DATA

Raw data:

animal;food elephant;vegetables tiger;meat

kangaroo;vegetables zebra;vegetables giraffe;vegetables

Or copy this in

```
zoo_eats = pd.DataFrame([['elephant','vegetables'], ['tiger','meat'],  
['kangaroo','vegetables'], ['zebra','vegetables'], ['giraffe','vegetables']],  
columns=['animal', 'food'])
```

string

integer

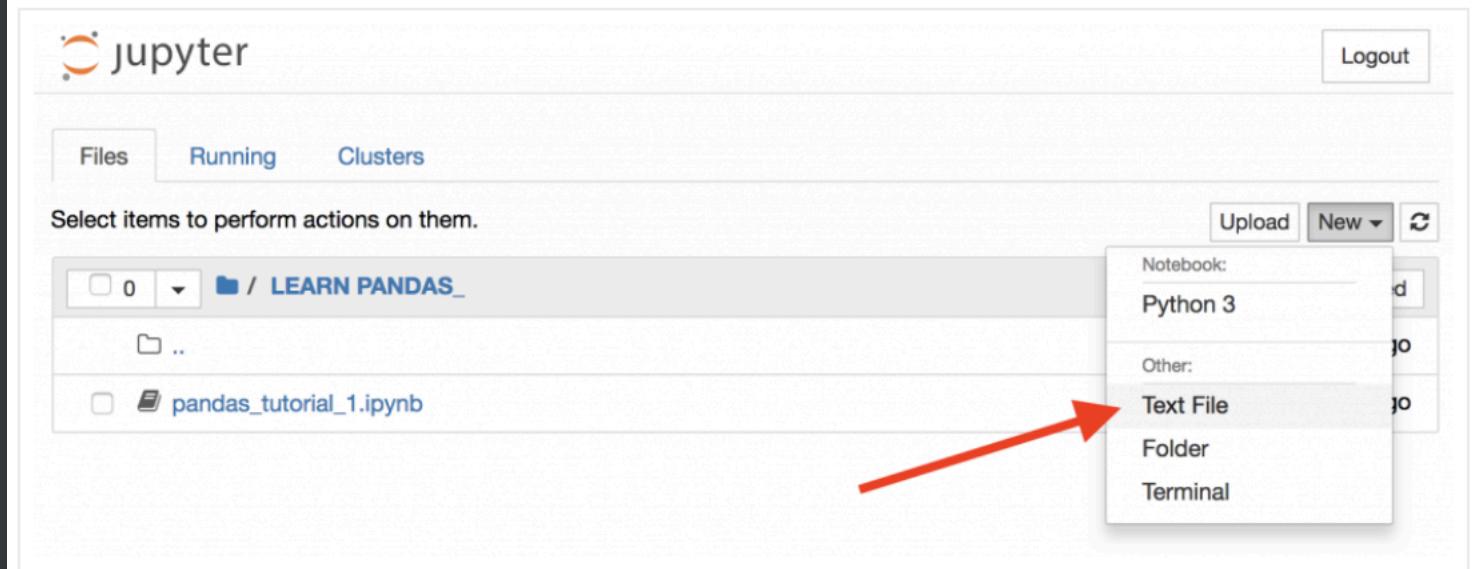
BUSINESS PROPOSAL

67



CREATE YOUR OWN CSV FILE

Go back to your Jupyter Home tab and create a new text file...



The image shows the Jupyter Home interface. At the top, there is a navigation bar with the Jupyter logo, a 'Logout' button, and tabs for 'Files', 'Running', and 'Clusters'. Below the navigation bar, a message says 'Select items to perform actions on them.' On the left, there is a file list showing a folder named 'LEARN PANDAS_' containing an 'ipynb' file named 'pandas_tutorial_1.ipynb'. On the right, a context menu is open with the following options: 'Upload', 'New', and a dropdown menu for 'Notebook' (set to 'Python 3'), 'Other' (set to 'Text File'), 'Folder', and 'Terminal'. A red arrow points to the 'Text File' option in the 'Other' dropdown.

jupyter

Logout

Files Running Clusters

Select items to perform actions on them.

0 / LEARN PANDAS_

pandas_tutorial_1.ipynb

Upload New

Notebook: Python 3

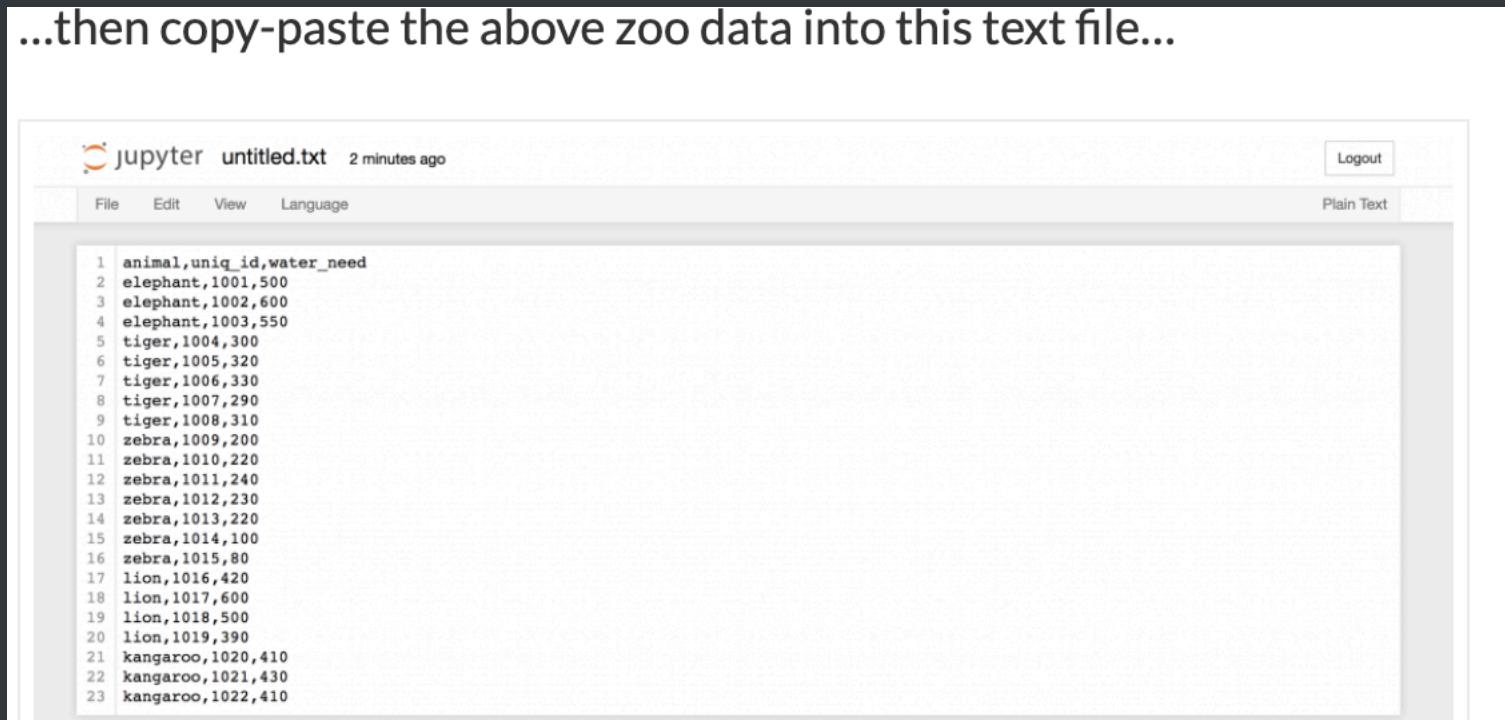
Other: Text File

Folder

Terminal

CREATE YOUR OWN CSV FILE

...then copy-paste the above zoo data into this text file...

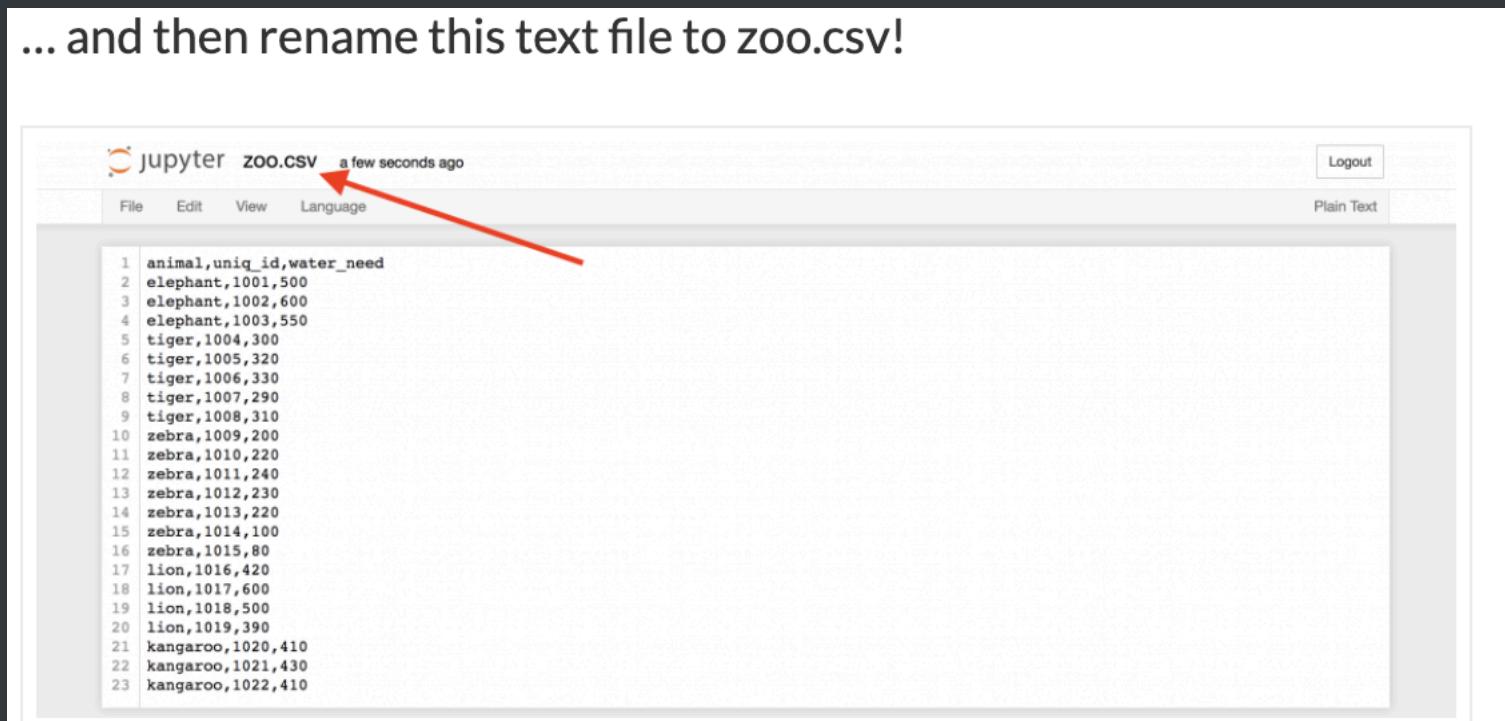


The screenshot shows a Jupyter Notebook interface with a single text cell. The cell title is 'untitled.txt' and it was created '2 minutes ago'. The cell content displays a CSV-like data structure with 23 rows, each containing an animal name, a unique ID, and a water need value. The data is as follows:

```
1 animal,uniq_id,water_need
2 elephant,1001,500
3 elephant,1002,600
4 elephant,1003,550
5 tiger,1004,300
6 tiger,1005,320
7 tiger,1006,330
8 tiger,1007,290
9 tiger,1008,310
10 zebra,1009,200
11 zebra,1010,220
12 zebra,1011,240
13 zebra,1012,230
14 zebra,1013,220
15 zebra,1014,100
16 zebra,1015,80
17 lion,1016,420
18 lion,1017,600
19 lion,1018,500
20 lion,1019,390
21 kangaroo,1020,410
22 kangaroo,1021,430
23 kangaroo,1022,410
```

CREATE YOUR OWN CSV FILE

... and then rename this text file to `zoo.csv`!



The screenshot shows a Jupyter Notebook interface. At the top, there is a header with the text "jupyter" and "zoo.csv" followed by "a few seconds ago". A red arrow points from the text "zoo.csv" to the file name in the header. Below the header is a menu bar with "File", "Edit", "View", and "Language" options. To the right of the menu bar are "Logout" and "Plain Text" buttons. The main content area displays a text file with the following content:

```
1 animal,uniq_id,water_need
2 elephant,1001,500
3 elephant,1002,600
4 elephant,1003,550
5 tiger,1004,300
6 tiger,1005,320
7 tiger,1006,330
8 tiger,1007,290
9 tiger,1008,310
10 zebra,1009,200
11 zebra,1010,220
12 zebra,1011,240
13 zebra,1012,230
14 zebra,1013,220
15 zebra,1014,100
16 zebra,1015,80
17 lion,1016,420
18 lion,1017,600
19 lion,1018,500
20 lion,1019,390
21 kangaroo,1020,410
22 kangaroo,1021,430
23 kangaroo,1022,410
```

MERGE

USE THE PANDAS MERGE METHOD

Zoo.merge(zoo_eats)

In [4]: `zoo.merge(zoo_eats)`

Out[4]:

	animal	uniq_id	water_need	food
0	elephant	1001	500	vegetables
1	elephant	1002	600	vegetables
2	elephant	1003	550	vegetables
3	tiger	1004	300	meat
4	tiger	1005	320	meat

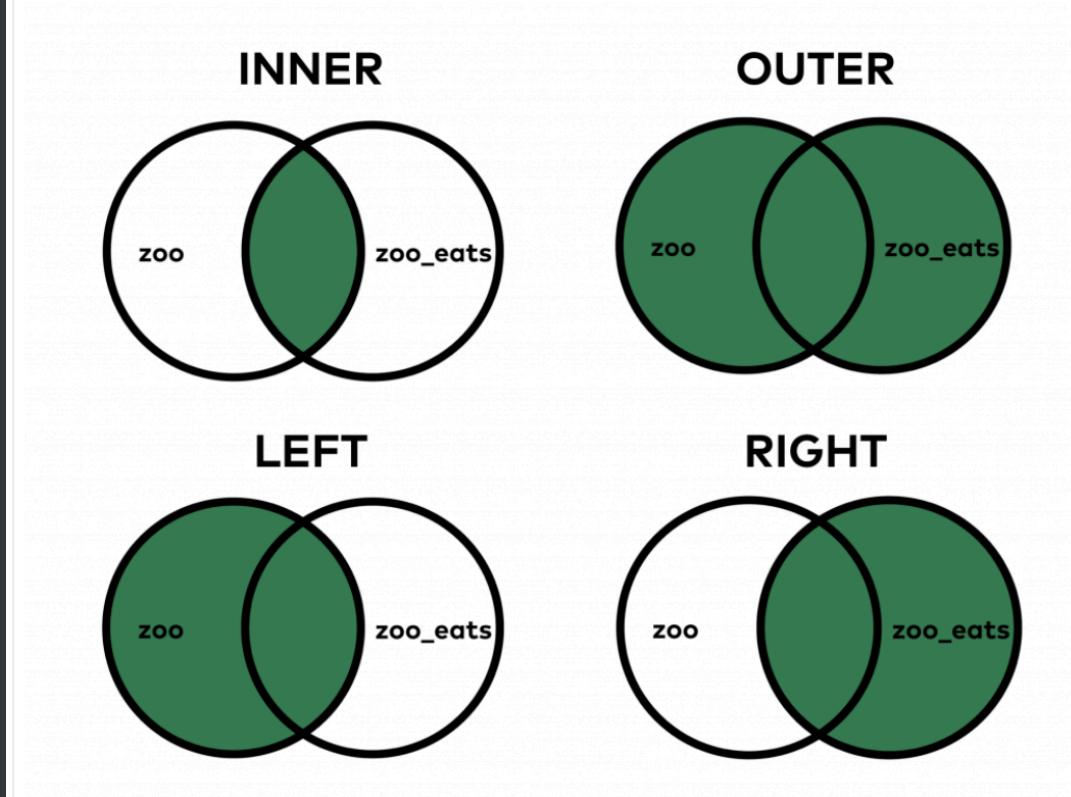
MERGE

WHAT JUST HAPPENED?

I just told python to merge `zoo` and `zoo_eats` together.

`Zoo_eats.merge(zoo)` is symmetric to `zoo.merge(zoo_eats)` – the only difference is the order of the columns in the output table.

DIFFERENT WAYS TO MERGE



OUTER MERGE

```
zoo.merge(zoo_eats, how = 'outer')
```

14	zebra	1015.0	80.0	vegetables
15	lion	1016.0	420.0	NaN
16	lion	1017.0	600.0	NaN
17	lion	1018.0	500.0	NaN
18	lion	1019.0	390.0	NaN
19	kangaroo	1020.0	410.0	vegetables
20	kangaroo	1021.0	430.0	vegetables
21	kangaroo	1022.0	410.0	vegetables
22	giraffe	NaN	NaN	vegetables

We have a NaN value in the columns where we didn't get information from other table.

LEFT MERGE

```
zoo.merge(zoo_eats, how = 'left')
```

15	lion	1016	420	NaN
16	lion	1017	600	NaN
17	lion	1018	500	NaN
18	lion	1019	390	NaN

Everything in zoo + what
things overlapping in zoo
eats

SETTING YOUR COLUMN

```
zoo.merge(zoo_eats, how = 'left', left_on = 'animal',  
right_on = 'animal')
```

Telling python what kind of columns are needed.

Most of the times, python finds these key columns

SORTING IN PANDAS

SORT_VALUES()

```
zoo.sort_values('water_need')
```

```
zoo.sort_values('water_need')
```

	animal	uniq_id	water_need
14	zebra	1015	80
13	zebra	1014	100
8	zebra	1009	200

SORTING IN PANDAS

SORT_VALUES()

```
zoo.sort_values(by = ['animal', 'water_need'])
```

```
zoo.sort_values(by = ['animal', 'water_need'])
```

	animal	uniq_id	water_need
0	elephant	1001	500
2	elephant	1003	550
1	elephant	1002	600
19	kangaroo	1020	410
21	kangaroo	1022	410

SORTING IN PANDAS

SORT_VALUES()
-
DESCENDING

```
zoo.sort_values(by = ['water_need'], ascending = False)
```

SETTING YOUR INDEX AFTER SORTING

AFTER SORTING,
YOUR INDEX IS MESSY WHICH CAN MAKE YOUR
VISUALIZATIONS WRONG

```
zoo.sort_values(by = ['water_need'], ascending =  
False).reset_index()
```

if you want to remove the old indexes, just add

```
zoo.sort_values(by = ['water_need'], ascending =  
False).reset_index(drop = True)
```

FILLNA

Fillna is fill+na.

After left merging, there are NaN values.

		2014	2015	2016	Vegetables
15	lion	1016	420	NaN	
16	lion	1017	600	NaN	
17	lion	1018	500	NaN	
18	lion	1019	390	NaN	

FILLNA

Now you can change all the fillna() function at once

```
zoo.merge(zoo_eats, how = 'left').fillna('unknown')
```

lion	1017	600	unknown
lion	1018	500	unknown
lion	1019	390	unknown

CHALLENGE

**OPEN YOUR PANDAS_TUTORIAL_READ CSV
DOWNLOAD ANOTHER DATASET BLOG_BUY BY RUNNING THE LINES IN
JUPYTER**

```
!wget 46.101.230.157/dilan/pandas_tutorial_buy.csv  
blog_buy = pd.read_csv('pandas_tutorial_buy.csv', delimiter=';', names =  
['my_date_time', 'event', 'user_id', 'amount'])
```

The article_read dataset shows all the users who read an article on the blog, and the blog_buy dataset shows all the users who bought something on the very same blog between 2018-01-01 and 2018-01-01

CHALLENGE

- **TASK #1:** What's the average (mean) revenue between 2018-01-01 and 2018-01-07 from the users in the article_read dataframe?
- **TASK #2:** Print the top 3 countries by total revenue between 2018-01-01 and 2018-01-07 ! (Obviously, this concerns the users in the article_read dataframe again.)

CHALLENGE

```
step_1 = article_read.merge(blog_buy, how = 'left', left_on = 'user_id', right_on = 'user_id')
step_2 = step_1.amount
step_3 = step_2.fillna(0)
result = step_3.mean()
result
```

SOLUTIONS

- (On the screenshot, at the beginning, I included the two extra cells where I import pandas and numpy, and where I read the csv files into my Jupyter Notebook.)
- In step_1, I merged the two tables (`article_read` and `blog_buy`) based on the `user_id` columns. I kept all the readers from `article_read` , even if they didn't buy anything, because `0` s should be counted in to the average revenue value. And I removed everyone who bought something but wasn't in the `article_read` dataset (that was fixed in the task). So all in all that led to a *left join*.
- In step_2, I removed all the unnecessary columns, and kept only `amount` .
- In step_3, I replaced `NaN` values with `0` s.
- And eventually I did the `.mean()` calculation.

SOLUTIONS

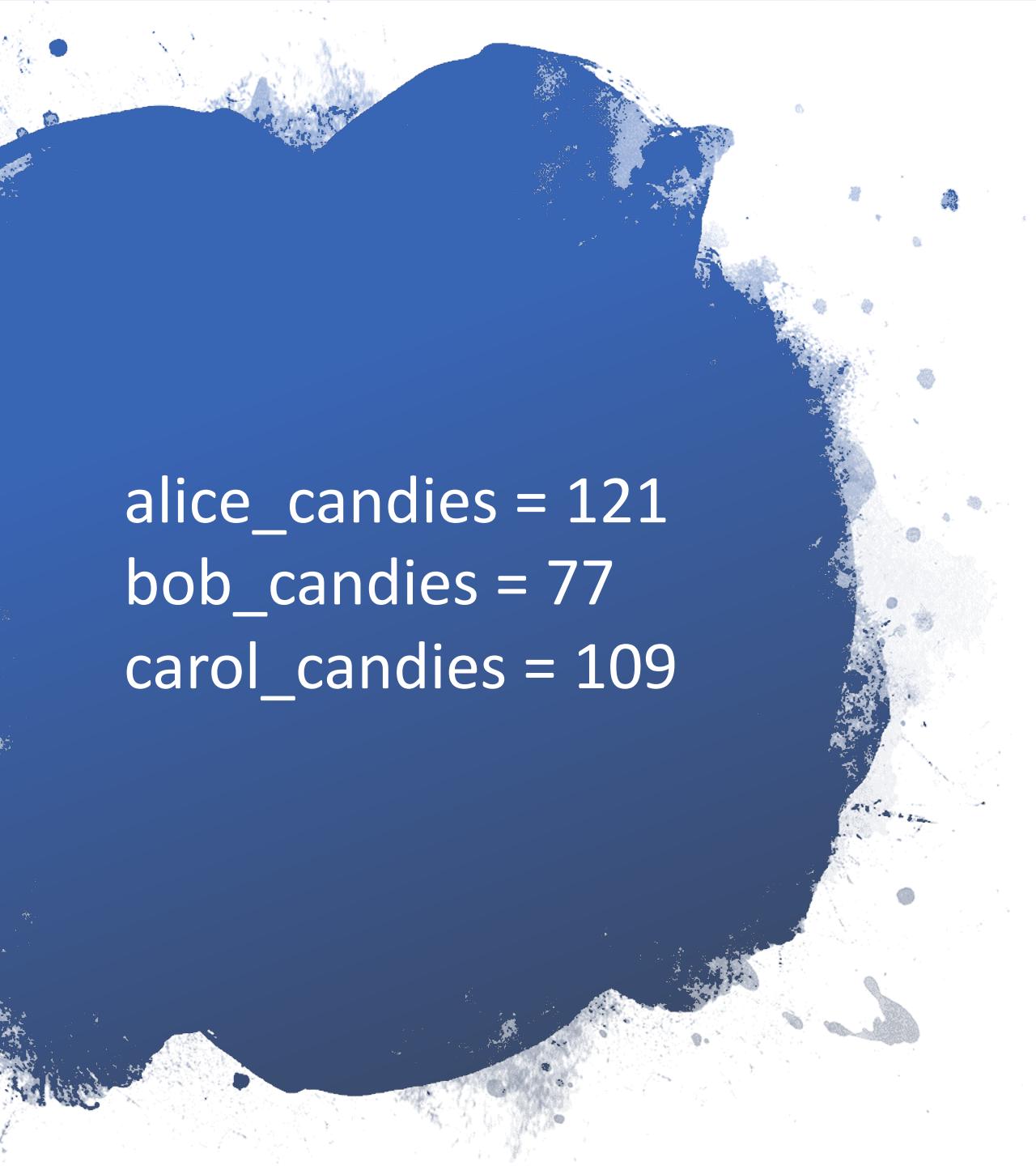
```
step_1 = article_read.merge(blog_buy, how = 'left', left_on = 'user_id', right_on = 'user_id')
step_2 = step_1.fillna(0)
step_3 = step_2.groupby('country').sum()
step_4 = step_3.amount
step_5 = step_4.sort_values(ascending = False)
step_5.head(3)
```

SOLUTIONS

- At **step_1**, I used the same merging method that I used in TASK #1.
- At **step_2**, I filled up all the `NaN` values with `0` s.
- At **step_3**, I summarized the numerical values by countries.
- At **step_4**, I took away all columns but `amount` .
- And at **step_5**, I sorted the results in descending order, so I can see my top list!
- Finally, I printed the first 3 lines only.

Datum

2019 Sep 18th



```
alice_candies = 121
bob_candies = 77
carol_candies = 109
```

Alice, Bob and Carol have agreed to pool their Halloween candy and split it evenly among themselves.

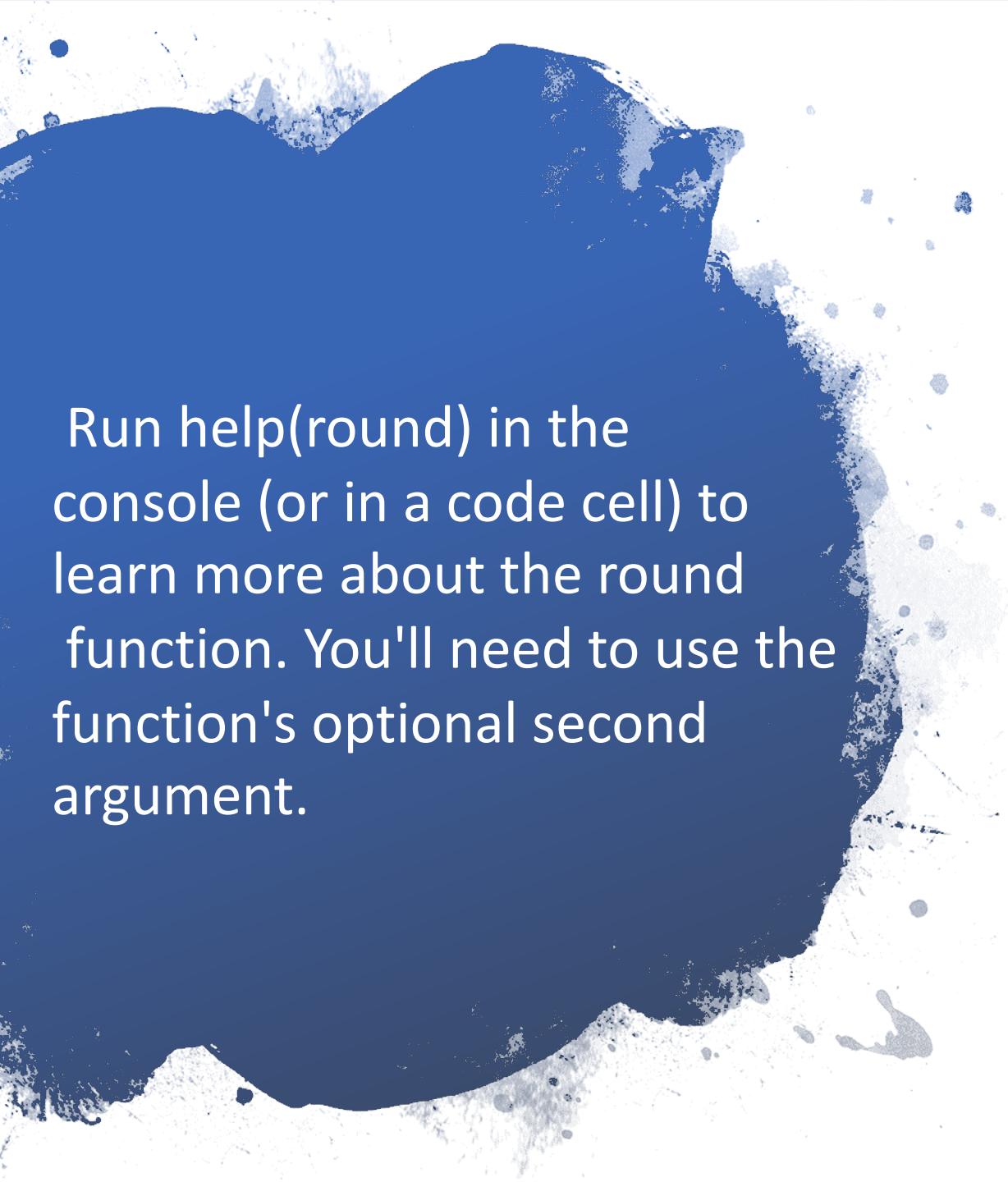
For the sake of their friendship, any candies left over will be smashed.

For example, if they collectively bring home 91 candies, they'll take 30 each and smash 1.

Write an arithmetic expression below to calculate how many candies they must smash for a given haul.

```
alice_candies = 121
bob_candies = 77
carol_candies = 109

to_smash = (alice_candies + bob_candies + carol_candies) % 3
print(to_smash)
```



Run `help(round)` in the console (or in a code cell) to learn more about the `round` function. You'll need to use the function's optional second argument.

Create a function that returns a given number rounded to two decimal places

```
>>> help(round)
```

```
Help on built-in function round in module builtins:
```

```
round(number, ndigits=None)
```

```
    Round a number to a given precision in decimal digits.
```

```
The return value is an integer if ndigits is omitted or None. Otherwise  
the return value has the same type as the number. ndigits may be negative.
```

```
>>> round(31.4432, 2)  
31.44
```



Define a function called `sign` which takes a numerical argument and returns -1 if it's negative, 1 if it's positive, and 0 if it's 0.

```
def sign(x):  
    if x > 0:  
        return 1  
    elif x < 0:  
        return -1  
    else:  
        return 0
```

For this exercise, we are using **Automobile Dataset**. This Automobile Dataset has a different characteristic of an auto such as body-style, wheel-base, engine-type, price, mileage, horsepower and many more.

What kind of things can we do with this data?
What kind of questions can you answer?

1. From given data set print first and last five row
2. Clean data and update the CSV file
 1. Replace all column values which contain with 0
 2. Find the most expensive car company name
 3. Print all Toyota car details
 4. Count total cars per company
 5. Find each company's highest price car
 6. Find the average mileage of each car making company
 7. Sort all cars by price column
 8. Merge data in two different ways
 9. Find the average in different columns

WHAT YOU SUGGESTED

DATUM SESSION

25th September

Booleans

Python has a type `bool` which can take on one of two values: `True` and `False`.

In [1]:

```
x = True
print(x)
print(type(x))
```

```
True
<class 'bool'>
```

Rather than putting `True` or `False` directly in our code, we usually get boolean values from **boolean operators**. These are operators that answer yes/no questions. We'll go through some of these operators below.

Comparison Operations

Operation	Description	Operation	Description
<code>a == b</code>	a equal to b	<code>a != b</code>	a not equal to b
<code>a < b</code>	a less than b	<code>a > b</code>	a greater than b
<code>a <= b</code>	a less than or equal to b	<code>a >= b</code>	a greater than or equal to b

In [2]:

```
def can_run_for_president(age):
    """Can someone of the given age run for president in the US?"""
    # The US Constitution says you must "have attained to the Age of thirty-five Years"
    return age >= 35

print("Can a 19-year-old run for president?", can_run_for_president(19))
print("Can a 45-year-old run for president?", can_run_for_president(45))
```

Can a 19-year-old run for president? False

Can a 45-year-old run for president? True

Exercise 1

Define a function that checks if the number is odd

Hint: Use the Modulus (%) function

Is 100 odd? False
Is -1 odd? True

My Answer:

```
1  def is_odd(n):  
2      return (n % 2) == 1  
3  
4  print("Is 100 odd?", is_odd(100))  
5  print("Is -1 odd?", is_odd(-1))  
6
```

Exercise 1

- Define a function that determines whether the number is positive, negative, zero.... Or something else

Hint: Use conditionals

0 is zero
-15 is negative

My Answer:

```
def inspect(x):
    if x == 0:
        print(x, "is zero")
    elif x > 0:
        print(x, "is positive")
    elif x < 0:
        print(x, "is negative")
    else:
        print(x, "is unlike anything I've ever seen...")

inspect(0)
inspect(-15)
```

Little spicier

- Calling `bool()` on an integer returns `False` if it's equal to 0 and `True` otherwise
- Using this, can you take advantage of this to write a succinct function that corresponds to the English sentence "does the customer want exactly one topping?"?

My Answer:

```
return (int(ketchup) + int(mustard) + int(onion)) == 1
#Fun fact: we don't technically need to call int on the arguments.
#Just by doing addition with booleans, Python implicitly does the integer conversion.
#So we could also write...

return (ketchup + mustard + onion) == 1
```

Lists

In [1]:

```
primes = [2, 3, 5, 7]
```

We can put other types of things in lists:

In [2]:

```
planets = ['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune']
```

We can even make a list of lists:

In [3]:

```
hands = [
    ['J', 'Q', 'K'],
    ['2', '2', '2'],
    ['6', 'A', 'K'], # (Comma after the last element is optional)
]
# (I could also have written this on one line, but it can get hard to read)
hands = [['J', 'Q', 'K'], ['2', '2', '2'], ['6', 'A', 'K']]
```

Indexing

Which planet is closest to the sun? Python uses *zero-based* indexing, so the first element has index 0.

In [5]:

```
planets[0]
```

Out[5]:

```
'Mercury'
```

What's the next closest planet?

In [6]:

```
planets[1]
```

Out[6]:

```
'Venus'
```

Which planet is *furthest* from the sun?

Elements at the end of the list can be accessed with negative numbers, starting from -1:

In [7]:

```
planets[-1]
```

Out[7]:

```
'Neptune'
```

In [8]:

```
planets[-2]
```

Out[8]:

```
'Uranus'
```

Slicing

What are the first three planets? We can answer this question using *slicing*:

In [9]:

```
planets[0:3]
```

Out[9]:

```
['Mercury', 'Venus', 'Earth']
```

`planets[0:3]` is our way of asking for the elements of `planets` starting from index 0 and continuing up to *but not including* index 3.

The starting and ending indices are both optional. If I leave out the start index, it's assumed to be 0. So I could rewrite the expression above as:

Slicing

```
In [11]: planets[3:]
```

```
Out[11]: ['Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune']
```

Negative Indices

In [12]:

```
# All the planets except the first and last
planets[1:-1]
```

Out[12]:

```
['Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranu
s']
```

In [13]:

```
# The last 3 planets
planets[-3:]
```

Out[13]:

```
['Saturn', 'Uranus', 'Neptune']
```

Mutable List

In [14]:

```
planets[3] = 'Malacandra'  
planets
```

Out[14]:

```
['Mercury',  
'Venus',  
'Earth',  
'Malacandra',  
'Jupiter',  
'Saturn',  
'Uranus',  
'Neptune']
```

Len()

`len` gives the length of a list:

In [16]:

```
# How many planets are there?  
len(planets)
```

Out[16]:

```
8
```

sorted()

In [17]:

```
# The planets sorted in alphabetical order
sorted(planets)
```

Out[17]:

```
['Earth', 'Jupiter', 'Mars', 'Mercury', 'Neptune', 'Satu
rn', 'Uranus', 'Venus']
```

Sum()

In [18]:

```
primes = [2, 3, 5, 7]
sum(primes)
```

Out[18]:

```
17
```

Lists

```
# Pluto is a planet darn it!
planets.append('Pluto')
```

```
planets.pop()
```

```
'Pluto'
```

Searching

```
]:
```

```
planets.index('Earth')
```

```
]:
```

```
2
```

```
.
```

```
# Is Earth a planet?
```

```
"Earth" in planets
```

```
:
```

```
True
```

Lists vs Tuples

1: The syntax for creating them uses parentheses instead of square brackets

```
|:
```

```
  t = (1, 2, 3)
```

Lists vs Tuples

2: They cannot be modified (they are *immutable*).

```
] : t[0] = 100
```

```
-----  
-----  
TypeError
```

```
  t recent call last)
```

```
  <ipython-input-36-e6cf7836e708> in <module>()
```

```
  ----> 1 t[0] = 100
```

```
Traceback (mos
```

```
TypeError: 'tuple' object does not support item assignme  
nt
```

Exercise

- The next iteration of Mario Kart will feature an extra-infuriating new item, the **Purple Shell**. When used, it warps the last place racer into first place and the first place racer into last place. Complete the function below to implement the Purple Shell's effect.

My Answer:

```
def purple_shell(racers):
    # One slick way to do the swap is x[0], x[-1] = x[-1], x[0].
    temp = racers[0]
    racers[0] = racers[-1]
    racers[-1] = temp
```

Spicy exercise

- We're using lists to record people who attended our party and what order they arrived in. For example, the following list represents a party with 7 guests, in which Adela showed up first and Ford was the last to arrive:
- `party_attendees = ['Adela', 'Fleda', 'Owen', 'May', 'Mona', 'Gilbert', 'Ford']`
- A guest is considered 'fashionably late' if they arrived after at least half of the party's guests. However, they must not be the very last guest (that's taking it too far). In the above example, Mona and Gilbert are the only guests who were fashionably late.
- Complete the function below which takes a list of party attendees as well as a person, and tells us whether that person is fashionably late.

My Answer:

Solution:

```
def fashionably_late(arrivals, name):
    order = arrivals.index(name)
    return order >= len(arrivals) / 2 and order != len(arrivals) -
```

The spiciest exercise

- <https://www.kaggle.com/theawy/exercise-booleans-and-conditionals/edit>
- Blackjack Problem!
Hint: you can follow the Blackjack strategy
(<https://en.wikipedia.org/wiki/Blackjack>) to re-design your logic

DATUM SESSION

6th November Review

Booleans

Python has a type `bool` which can take on one of two values: `True` and `False`.

In [1]:

```
x = True
print(x)
print(type(x))
```

```
True
<class 'bool'>
```

Rather than putting `True` or `False` directly in our code, we usually get boolean values from **boolean operators**. These are operators that answer yes/no questions. We'll go through some of these operators below.

Comparison Operations

Operation	Description	Operation	Description
<code>a == b</code>	a equal to b	<code>a != b</code>	a not equal to b
<code>a < b</code>	a less than b	<code>a > b</code>	a greater than b
<code>a <= b</code>	a less than or equal to b	<code>a >= b</code>	a greater than or equal to b

In [2]:

```
def can_run_for_president(age):
    """Can someone of the given age run for president in the US?"""
    # The US Constitution says you must "have attained to the Age of thirty-five Years"
    return age >= 35

print("Can a 19-year-old run for president?", can_run_for_president(19))
print("Can a 45-year-old run for president?", can_run_for_president(45))
```

Can a 19-year-old run for president? False

Can a 45-year-old run for president? True

Exercise 1

Define a function that checks if the number is odd

Hint: Use the Modulus (%) function

Is 100 odd? False
Is -1 odd? True

My Answer:

```
1  def is_odd(n):  
2      return (n % 2) == 1  
3  
4  print("Is 100 odd?", is_odd(100))  
5  print("Is -1 odd?", is_odd(-1))  
6
```

Exercise 1

- Define a function that determines whether the number is positive, negative, zero.... Or something else

Hint: Use conditionals

0 is zero
-15 is negative

My Answer:

```
def inspect(x):
    if x == 0:
        print(x, "is zero")
    elif x > 0:
        print(x, "is positive")
    elif x < 0:
        print(x, "is negative")
    else:
        print(x, "is unlike anything I've ever seen...")

inspect(0)
inspect(-15)
```

Lists

In [1]:

```
primes = [2, 3, 5, 7]
```

We can put other types of things in lists:

In [2]:

```
planets = ['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune']
```

We can even make a list of lists:

In [3]:

```
hands = [
    ['J', 'Q', 'K'],
    ['2', '2', '2'],
    ['6', 'A', 'K'], # (Comma after the last element is optional)
]
# (I could also have written this on one line, but it can get hard to read)
hands = [['J', 'Q', 'K'], ['2', '2', '2'], ['6', 'A', 'K']]
```

Indexing

Which planet is closest to the sun? Python uses *zero-based* indexing, so the first element has index 0.

In [5]:

```
planets[0]
```

Out[5]:

```
'Mercury'
```

What's the next closest planet?

In [6]:

```
planets[1]
```

Out[6]:

```
'Venus'
```

Which planet is *furthest* from the sun?

Elements at the end of the list can be accessed with negative numbers, starting from -1:

In [7]:

```
planets[-1]
```

Out[7]:

```
'Neptune'
```

In [8]:

```
planets[-2]
```

Out[8]:

```
'Uranus'
```

Slicing

What are the first three planets? We can answer this question using *slicing*:

In [9]:

```
planets[0:3]
```

Out[9]:

```
['Mercury', 'Venus', 'Earth']
```

`planets[0:3]` is our way of asking for the elements of `planets` starting from index 0 and continuing up to *but not including* index 3.

The starting and ending indices are both optional. If I leave out the start index, it's assumed to be 0. So I could rewrite the expression above as:

Negative Indices

In [12]:

```
# All the planets except the first and last
planets[1:-1]
```

Out[12]:

```
['Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranu
s']
```

In [13]:

```
# The last 3 planets
planets[-3:]
```

Out[13]:

```
['Saturn', 'Uranus', 'Neptune']
```

Mutable List

In [14]:

```
planets[3] = 'Malacandra'  
planets
```

Out[14]:

```
['Mercury',  
'Venus',  
'Earth',  
'Malacandra',  
'Jupiter',  
'Saturn',  
'Uranus',  
'Neptune']
```

Len()

`len` gives the length of a list:

In [16]:

```
# How many planets are there?  
len(planets)
```

Out[16]:

```
8
```

Slicing

```
In [11]: planets[3:]
```

```
Out[11]: ['Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune']
```

sorted()

In [17]:

```
# The planets sorted in alphabetical order
sorted(planets)
```

Out[17]:

```
['Earth', 'Jupiter', 'Mars', 'Mercury', 'Neptune', 'Satu
rn', 'Uranus', 'Venus']
```

Sum()

In [18]:

```
primes = [2, 3, 5, 7]
sum(primes)
```

Out[18]:

```
17
```

Lists

```
# Pluto is a planet darn it!
planets.append('Pluto')
```

```
planets.pop()
```

```
'Pluto'
```

Searching

```
]:
```

```
planets.index('Earth')
```

```
]:
```

```
2
```

```
.
```

```
# Is Earth a planet?
```

```
"Earth" in planets
```

```
:
```

```
True
```

Lists vs Tuples

1: The syntax for creating them uses parentheses instead of square brackets

```
|:
```

```
  t = (1, 2, 3)
```

Lists vs Tuples

2: They cannot be modified (they are *immutable*).

```
] : t[0] = 100
```

```
-----  
-----  
TypeError  
  t recent call last)  
<ipython-input-36-e6cf7836e708> in <module>()  
----> 1 t[0] = 100
```

Traceback (mos

```
TypeError: 'tuple' object does not support item assignment
```

Exercise

- The next iteration of Mario Kart will feature an extra-infuriating new item, the **Purple Shell**. When used, it warps the last place racer into first place and the first place racer into last place. Complete the function below to implement the Purple Shell's effect.

My Answer:

```
def purple_shell(racers):
    # One slick way to do the swap is x[0], x[-1] = x[-1], x[0].
    temp = racers[0]
    racers[0] = racers[-1]
    racers[-1] = temp
```

5.

We're using lists to record people who attended our party and what order they arrived in. For example, the following list represents a party with 7 guests, in which Adela showed up first and Ford was the last to arrive:

```
party_attendees = ['Adela', 'Fleda', 'Owen', 'May', 'Mona', 'Gilbert', 'Ford']
```

A guest is considered 'fashionably late' if they arrived after at least half of the party's guests. However, they must not be the very last guest (that's taking it too far). In the above example, Mona and Gilbert are the only guests who were fashionably late.

Complete the function below which takes a list of party attendees as well as a person, and tells us whether that person is fashionably late.

In[]:

```
def fashionably_late(arrivals, name):
    """Given an ordered list of arrivals to the party and a name, return whether the guest with that
    name was fashionably late.
    """
    pass
```

```
q5.check()
```

My Answer:

Solution:

```
def fashionably_late(arrivals, name):
    order = arrivals.index(name)
    return order >= len(arrivals) / 2 and order != len(arrivals) -
```

Strings

Nov 6th

String syntax

You've already seen plenty of strings in examples during the previous lessons, but just to recap, strings in Python can be defined using either single or double quotations. They are functionally equivalent.

In [1]:

```
x = 'Pluto is a planet'  
y = "Pluto is a planet"  
x == y
```

Out[1]:

```
True
```

In [2]:

```
print("Pluto's a planet!")
print('My dog is named "Pluto"')
```

```
Pluto's a planet!
My dog is named "Pluto"
```

If we try to put a single quote character inside a single-quoted string, Python gets confused:

In [3]:

```
'Pluto's a planet!'
```

```
File "<ipython-input-3-a43631749f52>", line 1
  'Pluto's a planet!'
          ^
SyntaxError: invalid syntax
```

We can fix this by "escaping" the single quote with a backslash.

In [4]:

```
'Pluto\'s a planet!'
```

Out[4]:

```
"Pluto's a planet!"
```

Exercise

Make output look like this using the print function:

What's up?

That's "cool"

Look, a mountain: /\

What you type...	What you get	example	print(example)
\'	'	'What\'s up?'	What's up?
\"	"	"That's \"cool\""	That's "cool"
\\"	\	"Look, a mountain: /\\"	Look, a mountain: /\

The last sequence, `\n`, represents the *newline character*. It causes Python to start a new line.

In [5]:

```
hello = "hello\nworld"  
print(hello)
```

```
hello  
world
```

In addition, Python's triple quote syntax for strings lets us include newlines literally (i.e. by just hitting 'Enter' on our keyboard, rather than using the special '\n' sequence). We've already seen this in the docstrings we use to document our functions, but we can use them anywhere we want to define a string.

In [6]:

```
triplequoted_hello = """hello
world"""
print(triplequoted_hello)
triplequoted_hello == hello
```

```
hello
world
```

Out[6]:

```
True
```

The `print()` function automatically adds a newline character unless we specify a value for the keyword argument `end` other than the default value of `'\n'`:

In [7]:

```
print("hello")
print("world")
print("hello", end=' ')
print("pluto", end=' ')
```

```
hello
world
hellopluto
```

Strings are sequences

Strings can be thought of as sequences of characters. Almost everything we've seen that we can do to a list, we can also do to a string.

In [8]:

```
# Indexing
planet = 'Pluto'
planet[0]
```

Out[8]:

```
'P'
```

In [9]:

```
# Slicing
planet[-3:]
```

Out[9]:

```
'uto'
```

In [10]:

```
# How long is this string?  
len(planet)
```

Out[10]:

```
5
```

In [11]:

```
# Yes, we can even loop over them  
[char+'!' for char in planet]
```

Out[11]:

```
['P! ', 'l! ', 'u! ', 't! ', 'o! ']
```

String methods

Like `list`, the type `str` has lots of very useful methods. I'll show just a few examples here.

In [13]:

```
# ALL CAPS
claim = "Pluto is a planet!"
claim.upper()
```

Out[13]:

```
'PLUTO IS A PLANET!'
```

In [14]:

```
# all lowercase
claim.lower()
```

Out[14]:

```
'pluto is a planet!'
```

In [15]:

```
# Searching for the first index of a substring
claim.index('plan')
```

Out[15]:

```
11
```

In [16]:

```
claim.startswith(planet)
```

Out[16]:

```
True
```

In [17]:

```
claim.endswith('dwarf planet')
```

Out[17]:

```
False
```

Going between strings and lists: `.split()` and `.join()`

`str.split()` turns a string into a list of smaller strings, breaking on whitespace by default. This is super useful for taking you from one big string to a list of words.

In [18]:

```
words = claim.split()  
words
```

Out[18]:

```
['Pluto', 'is', 'a', 'planet!']
```

Occasionally you'll want to split on something other than whitespace:

In [19]:

```
datestr = '1956-01-31'  
year, month, day = datestr.split('-')
```

In [20]:

```
' / '.join([month, day, year])
```

Out[20]:

```
'01/31/1956'
```

In [21]:

```
# Yes, we can put unicode characters right in our string literals :)  
'手掌'.join([word.upper() for word in words])
```

Out[21]:

```
'PLUTO 🌌 IS 🌌 A 🌌 PLANET!'
```

Building strings with `.format()`

Python lets us concatenate strings with the `+` operator.

In [22]:

```
planet + ', we miss you.'
```

Out[22]:

```
'Pluto, we miss you.'
```

If we want to throw in any non-string objects, we have to be careful to call `str()` on them first

In [23]:

```
position = 9
planet + ", you'll always be the " + position + "th planet to me."
```

```
-----
TypeError                                 Traceback (most recent call last)
<ipython-input-23-73295f9638cc> in <module>
      1 position = 9
----> 2 planet + ", you'll always be the " + position + "th planet to me."
      3
TypeError: must be str, not int
```

In [24]:

```
planet + ", you'll always be the " + str(position) + "th planet to me."
```

Out[24]:

```
"Pluto, you'll always be the 9th planet to me."
```

This is getting hard to read and annoying to type. `str.format()` to the rescue.

In [25]:

```
"{}, you'll always be the {}th planet to me.".format(planet, position)
```

Out[25]:

```
"Pluto, you'll always be the 9th planet to me."
```

So much cleaner! We call `.format()` on a "format string", where the Python values we want to insert are represented with `{}` placeholders.

Notice how we didn't even have to call `str()` to convert `position` from an int. `format()` takes care of that for us.

If that was all that `format()` did, it would still be incredibly useful. But as it turns out, it can do a *lot* more. Here's just a taste:

In [26]:

```
pluto_mass = 1.303 * 10**22
earth_mass = 5.9722 * 10**24
population = 52910390
#           2 decimal points  3 decimal points, format as percent      separate with commas
"{} weighs about {:.2} kilograms ( {:.3%} of Earth's mass). It is home to {:,} Plutonians.".format(
    planet, pluto_mass, pluto_mass / earth_mass, population,
)
```

Out[26]:

```
"Pluto weighs about 1.3e+22 kilograms (0.218% of Earth's mass). It is home to 52,910,390 Plutonians."
```

In [27]:

```
# Referring to format() arguments by index, starting from 0
s = """Pluto's a {0}.
No, it's a {1}.
{0}!
{1}!""".format('planet', 'dwarf planet')
print(s)
```

```
Pluto's a planet.
No, it's a dwarf planet.
planet!
dwarf planet!
```

Exercise

Hint: Try looking up `help(str.isdigit)`

1.

There is a saying that "Data scientists spend 80% of their time cleaning data, and 20% of their time complaining about cleaning data." Let's see if you can write a function to help clean US zip code data. Given a string, it should return whether or not that string represents a valid zip code. For our purposes, a valid zip code is any string consisting of exactly 5 digits.

HINT: `str` has a method that will be useful here. Use `help(str)` to review a list of string methods.

In[7]:

```
def is_valid_zip(zip_code):
    """Returns whether the input string is a valid (5 digit) zip code
    """
    pass

q1.check()
```



```
1  def is_valid_zip(zip_str):  
2      return len(zip_str) == 5 and zip_str.isdigit()
```

`isdigit(self, /)`

Return True if the string is a digit string, False otherwise.

A string is a digit string if all characters in the string are digits and there is at least one character in the string.

Dictionaries

Dictionaries

Dictionaries are a built-in Python data structure for mapping keys to values.

In [28]:

```
numbers = {'one':1, 'two':2, 'three':3}
```

In this case `'one'`, `'two'`, and `'three'` are the **keys**, and 1, 2 and 3 are their corresponding values.

Values are accessed via square bracket syntax similar to indexing into lists and strings.

In [29]:

```
numbers['one']
```

Out[29]:

```
1
```

We can use the same syntax to add another key, value pair

In [30]:

```
numbers['eleven'] = 11
numbers
```

Out[30]:

```
{'one': 1, 'two': 2, 'three': 3, 'eleven': 11}
```

Or to change the value associated with an existing key

In [31]:

```
numbers['one'] = 'Pluto'
numbers
```

Out[31]:

```
{'one': 'Pluto', 'two': 2, 'three': 3, 'eleven': 11}
```

Python has *dictionary comprehensions* with a syntax similar to the list comprehensions we saw in the previous tutorial.

In [32]:

```
planets = ['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune']
planet_to_initial = {planet: planet[0] for planet in planets}
planet_to_initial
```

Out[32]:

```
{'Mercury': 'M',
 'Venus': 'V',
 'Earth': 'E',
 'Mars': 'M',
 'Jupiter': 'J',
 'Saturn': 'S',
 'Uranus': 'U',
 'Neptune': 'N'}
```

The `in` operator tells us whether something is a key in the dictionary

In [33]:

```
'Saturn' in planet_to_initial
```

Out[33]:

```
True
```

In [34]:

```
'Betelgeuse' in planet_to_initial
```

Out[34]:

```
False
```

A for loop over a dictionary will loop over its keys

In [35]:

```
for k in numbers:  
    print("{} = {}".format(k, numbers[k]))
```

```
one = Pluto  
two = 2  
three = 3  
eleven = 11
```

We can access a collection of all the keys or all the values with `dict.keys()` and `dict.values()`, respectively.

In [36]:

```
# Get all the initials, sort them alphabetically, and put them in a space-separated string.  
''.join(sorted(planet_to_initial.values()))
```

Out[36]:

```
'E J M M N S U V'
```

The very useful `dict.items()` method lets us iterate over the keys and values of a dictionary simultaneously. (In Python jargon, an **item** refers to a key, value pair)

In [37]:

```
for planet, initial in planet_to_initial.items():
    print("{} begins with \"{}\"".format(planet.rjust(10), initial))
```

```
Mercury begins with "M"
Venus begins with "V"
Earth begins with "E"
Mars begins with "M"
Jupiter begins with "J"
Saturn begins with "S"
Uranus begins with "U"
Neptune begins with "N"
```

2.

A researcher has gathered thousands of news articles. But she wants to focus her attention on articles including a specific word. Complete the function below to help her filter her list of articles.

Your function should meet the following criteria

- Do not include documents where the keyword string shows up only as a part of a larger word. For example, if she were looking for the keyword “closed”, you would not include the string “enclosed.”
- She does not want you to distinguish upper case from lower case letters. So the phrase “Closed the case.” would be included when the keyword is “closed”
- Do not let periods or commas affect what is matched. “It is closed.” would be included when the keyword is “closed”. But you can assume there are no other types of punctuation.

```
def word_search(doc_list, keyword):
    """
    Takes a list of documents (each document is a string) and a keyword.
    Returns list of the index values into the original list for all documents
    containing the keyword.

    Example:
    doc_list = ["The Learn Python Challenge Casino.", "They bought a car", "Casinoville"]
    >>> word_search(doc_list, 'casino')
    >>> [0]
    """
    pass
```

Some methods that may be useful here:
str.split()
str.strip()
str.lower()

Solution:

```
def word_search(documents, keyword):
    # list to hold the indices of matching documents
    indices = []
    # Iterate through the indices (i) and elements (doc) of documents
    for i, doc in enumerate(documents):
        # Split the string doc into a list of words (according to whitespace)
        tokens = doc.split()
        # Make a transformed list where we 'normalize' each word to facilitate matching.
        # Periods and commas are removed from the end of each word, and it's set to all lowercase.
        normalized = [token.rstrip('.,').lower() for token in tokens]
        # Is there a match? If so, update the list of matching indices.
        if keyword.lower() in normalized:
            indices.append(i)
    return indices
```

```
In [1]: def word_search(documents, keyword):
    # list to hold the indices of matching documents
    indices = []
    # Iterate through the indices (i) and elements (doc) of documents
    for i, doc in enumerate(documents):
        # Split the string doc into a list of words (according to whitespace)
        tokens = doc.split()
        # Make a transformed list where we 'normalize' each word to facilitate matching.
        # Periods and commas are removed from the end of each word, and it's set to all lowercase.
        normalized = [token.rstrip('.').lower() for token in tokens]
        # Is there a match? If so, update the list of matching indices.
        if keyword.lower() in normalized:
            indices.append(i)
    return indices
```

```
In [14]: word_search(["The Learn Python Challenge Casino.", "They bought a car", "Casinoville"], "casino")
Out[14]: [0]
```