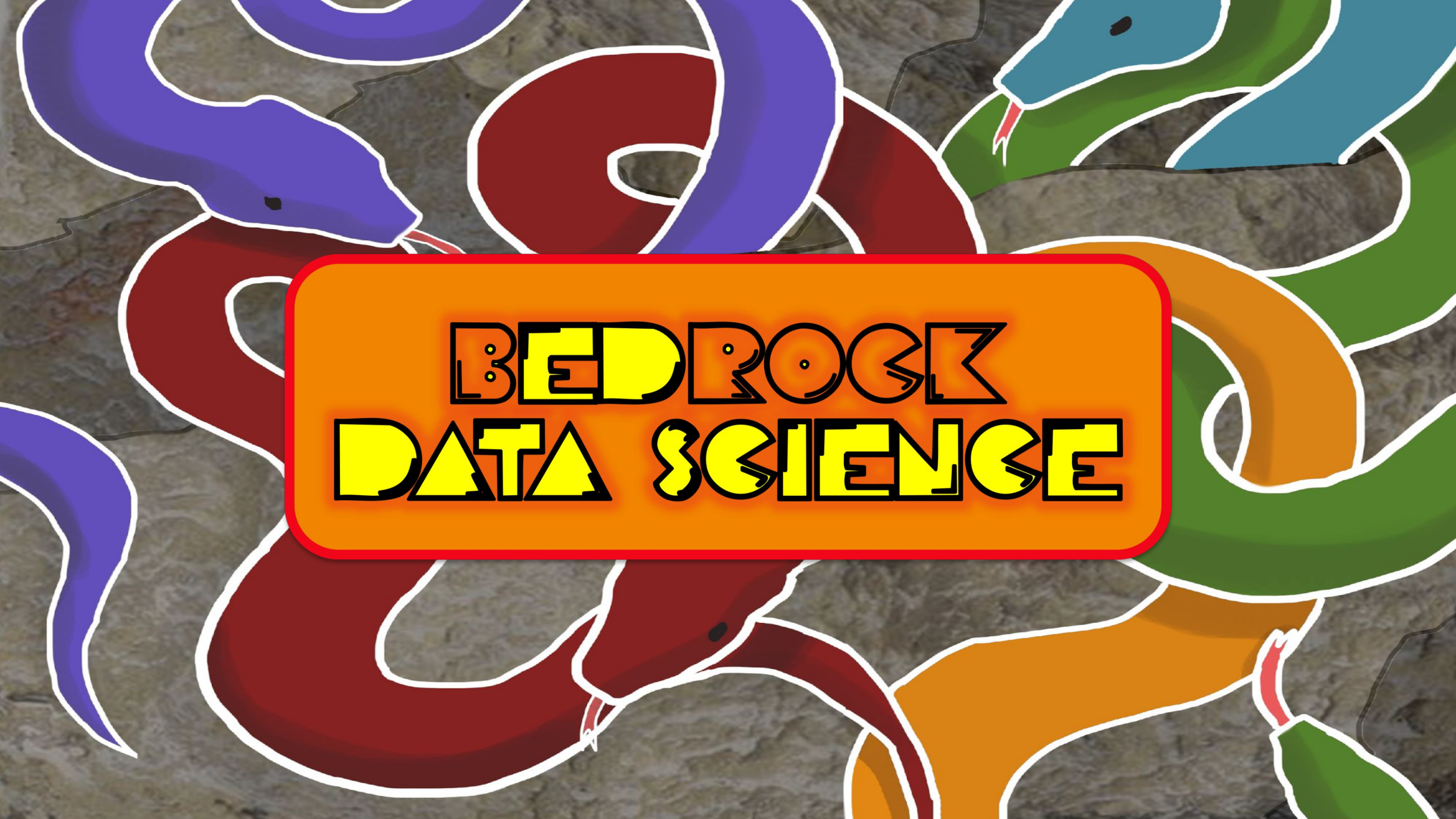
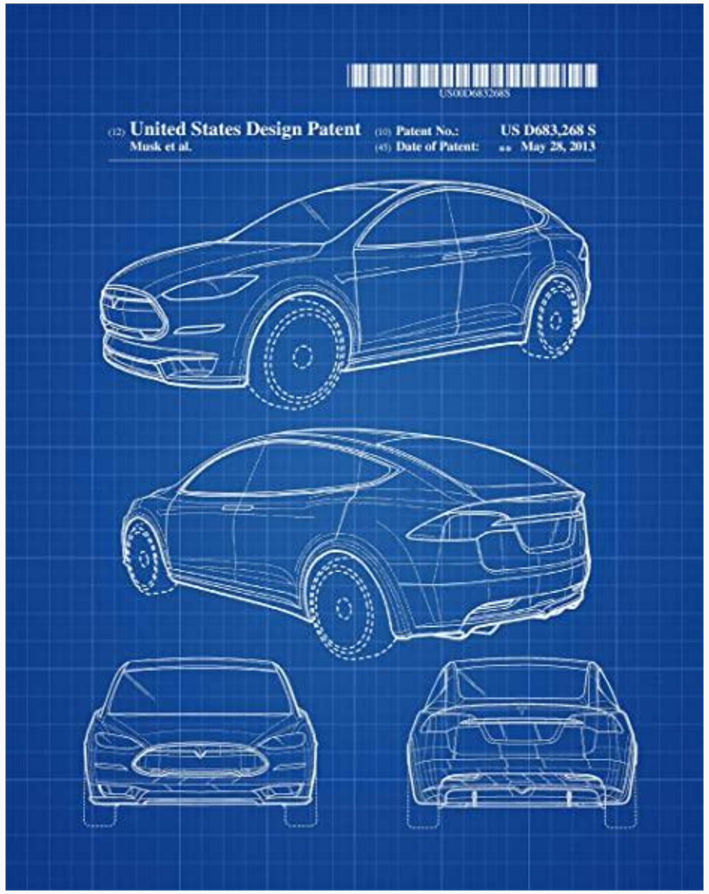


BEDROCK DATA SCIENCE

Previously on Bedrock Data Science

RECAP: Classes

The class template is often called a "blueprint"


United States Design Patent
Musk et al.
Patent No.: US D683,268 S
Date of Patent: May 28, 2013

We instantiate the class object instance with function notation

RECAP: Dunder methods

1. Dunder methods I
   1. __init__()
   2. __call__()
2. Dunder methods II
   1. __setitem__()
   2. __getitem__()
3. Dunder methods III
   1. __repr__()
   2. __str__()

```
from random import randrange as rand

class Harlist_subclass(Harlist):
    def __getitem__(self, key):
        return self.list[key]
    def __setitem__(self, index, value):
        self.list[index] = value

# Create a class instance
>>> random_ints = Harlist_ subclass()
>>> random_ints.list
[4, 3, 4, 2, 4]
>>> random_ints[0]
4
>>> random_ints[0] = 100
>>> random_ints[0]
100
>>> print(random_int)
<__main__.Rand_Int_List object at 0x7fb8d029b700>
```

RECAP: Classes

I. Encapsulation

II. Abstraction

III. Inheritance

IV. Polymorphism

STRINGS

String Literals

What is a python string ?

"Hello, World!"

'83110 70510'

" 👋 🌍
"

String Literals

- Collection of characters (including emojis).
- Starts and ends with either double quotes or single quotes or even triple quotes.
- Like tuples, it can't be modified after initialization.

✓ ✓ ✓

```
option_1 = 'Hello, World!'  option_2 = "Hello, World!"      option_3 = '''
                                                            ........  Hello,
                                                            ........  World!
                                                                       '''
```

String Literals

- Collection of characters (including emojis)
- Starts and ends with either double quotes or single quotes or even triple quotes
- Like tuples, it can't be modified after initialization

✓

```
option_1 = 'Hello, World!'
```

✓

```
option_2 = "Hello, World!"
```

✓

```
option_3 = '''
........  Hello,
........  World!
           '''
```

Tripping over quotes

```
In [13]: print('Say hi to my 'friend'')
  File "<ipython-input-13-7108e0539fd5>", line 1
    print('Say hi to my 'friend'')
                              ^
SyntaxError: invalid syntax
```

```
print('Say hi to my "friend"')
>>> Say hi to my "friend"
```

backslash (\) is an escape character. We will see more of it in later sessions

```
print('Say hi to my \'friend\'')
>>> Say hi to my 'friend'
```

String operations

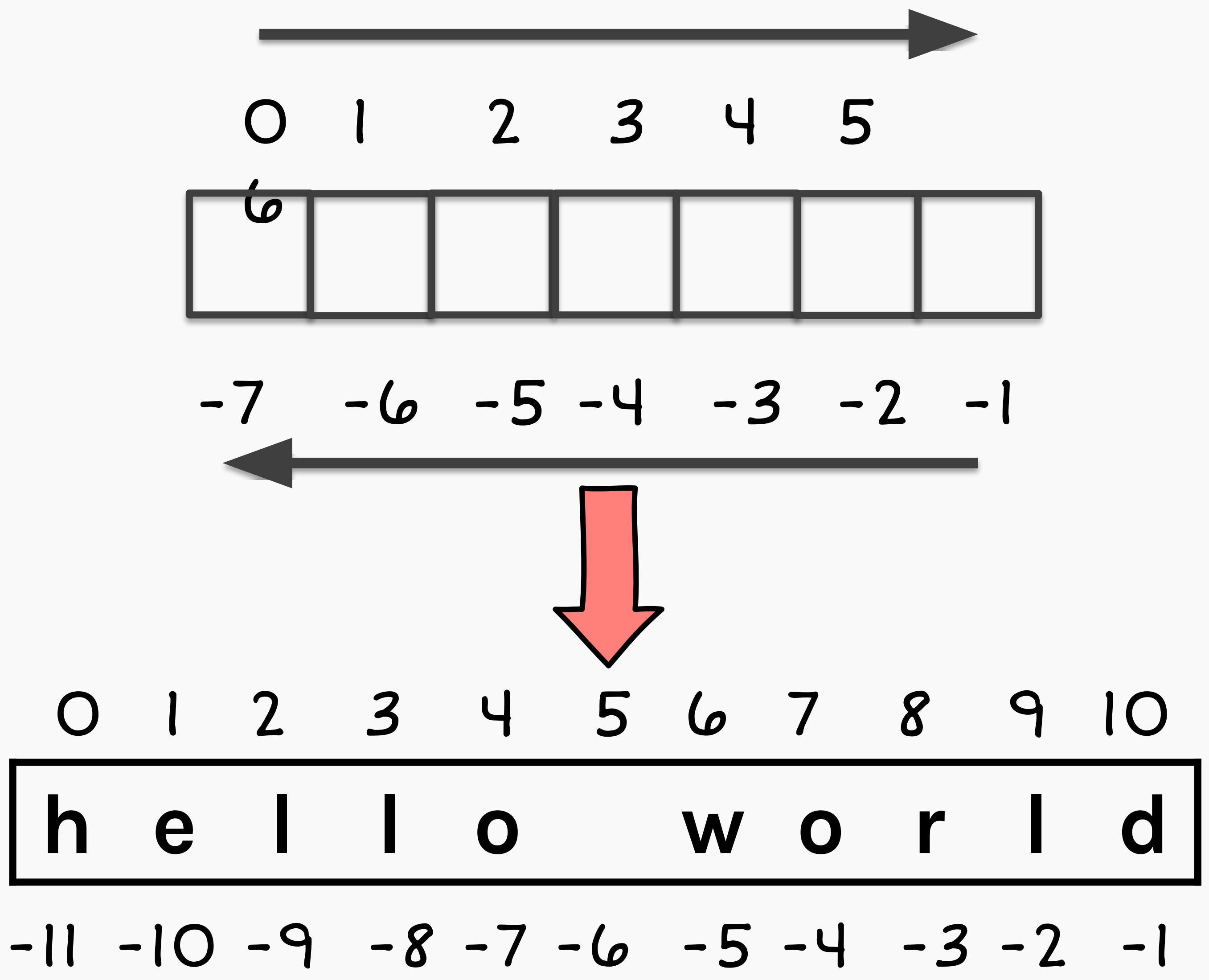
"Hello" + " World!" "Hello World!"

The **in** operator can be used to check for an exact match of a string in another string

```
"Hello" in "Hello World!"
>>> True
"hello" in "Hello World!"
>>> False
```

Accessing a string

- Strings can be indexed just like lists.
- Using this we can use them in loops



For loop

Using a for loop with a string

```
for char in 'Hello, World':
    print(char, end="")
>>> Hello, World
```

For loop

```
for char in 'Hello, World':
    print(char)
>>>H
>>>e
>>>l
>>>l
>>>o
>>>,
>>>
>>>...
```

String methods

'capitalize()'

'center()'

'count()'

'endswith()'

'find()'

'format()'

'replace()'

'index()'

'isalnum()'

'isalpha()'

'isdecimal()'

'isdigit()'

'lower()'

'lstrip()'

'islower()'

'isnumeric()'

'isspace()'

'istitle()'

'isupper()'

'join()'

'ljust()'

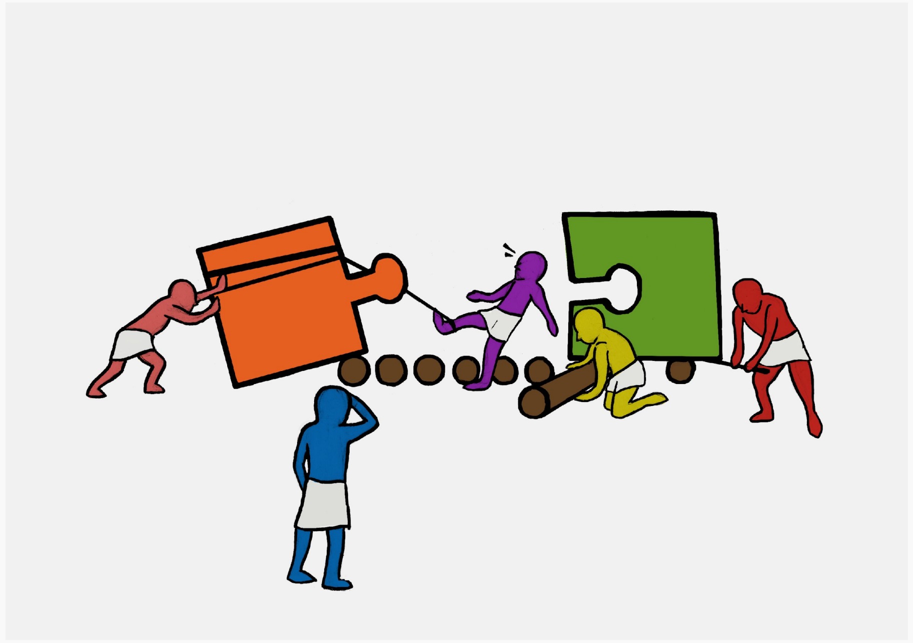
And a lot more …

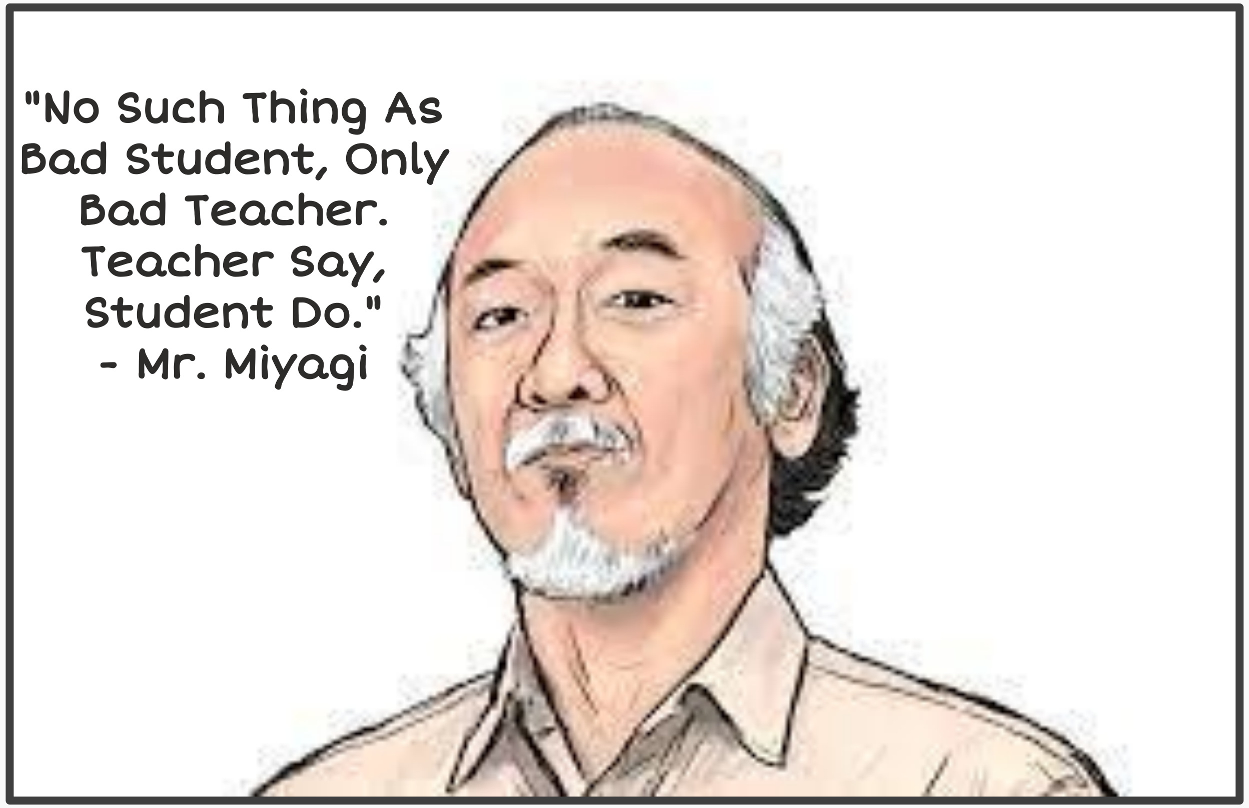
String methods

```
'capitalize()'      'index()'        'islower()'
'center()'          'isalnum()'      'isnumeric()'
'count()'           'isalpha()'      'isspace()'
'endswith()'        'isdecimal()'    'istitle()'
'find()'            'isdigit()'      'isupper()'
'format()'          'lower()'        'join()'
'replace()'         'lstrip()'       'ljust()'
```

And a lot more …

Exercise #1




"No Such Thing As Bad Student, Only Bad Teacher. Teacher Say, Student Do."
- Mr. Miyagi

Files

File formats

.txt	Text Data
.csv	Tabular Data
.png	Image data
.mp4	Video data

File formats

.txt	Text Data
.csv	Tabular Data
.png	Image data
.mp4	Video data

Text files

```
f = open('file.txt')
text = f.read()
f.close()
```

Text files

```
f = open('file.txt')
text = f.read()
f.close()
```

Text files

```
f = open('file.txt')
text = f.read()
f.close()
```

It is important to close files because when writing to a file, the data may not be written to disk until the file is closed.

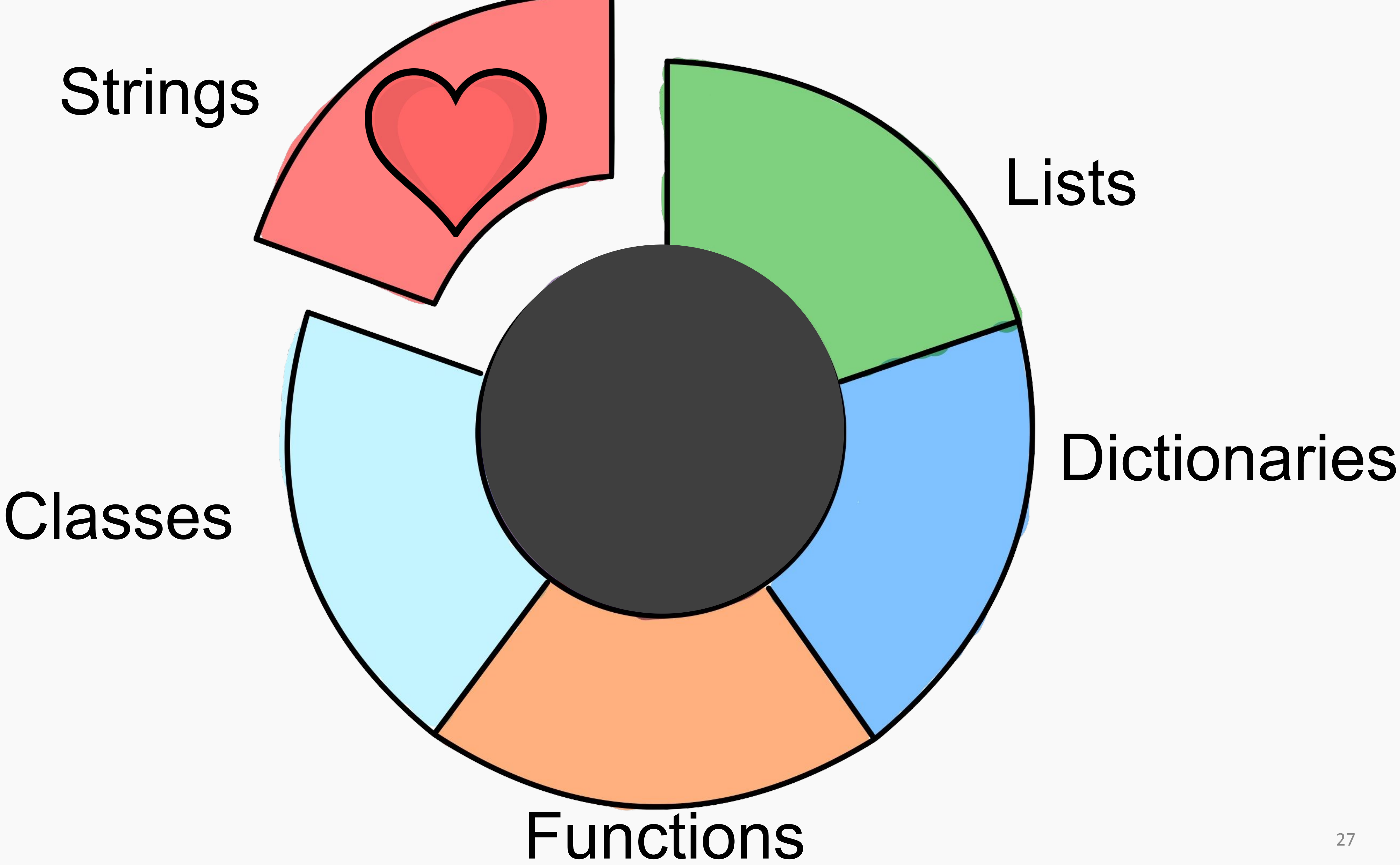
THE END

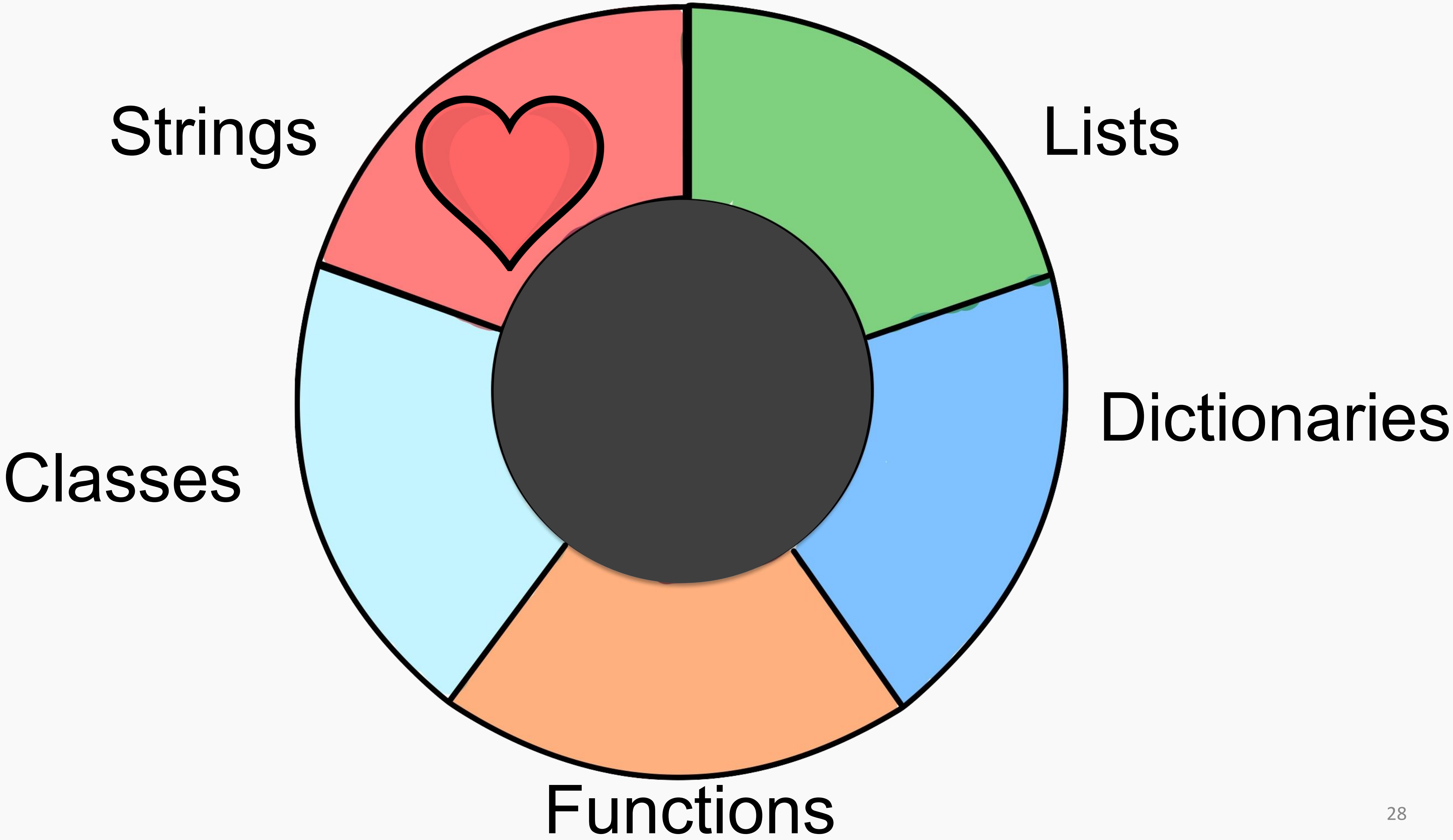
.txt files in 2 lines

Using the **with** keyword, we can open files and not worry about closing them

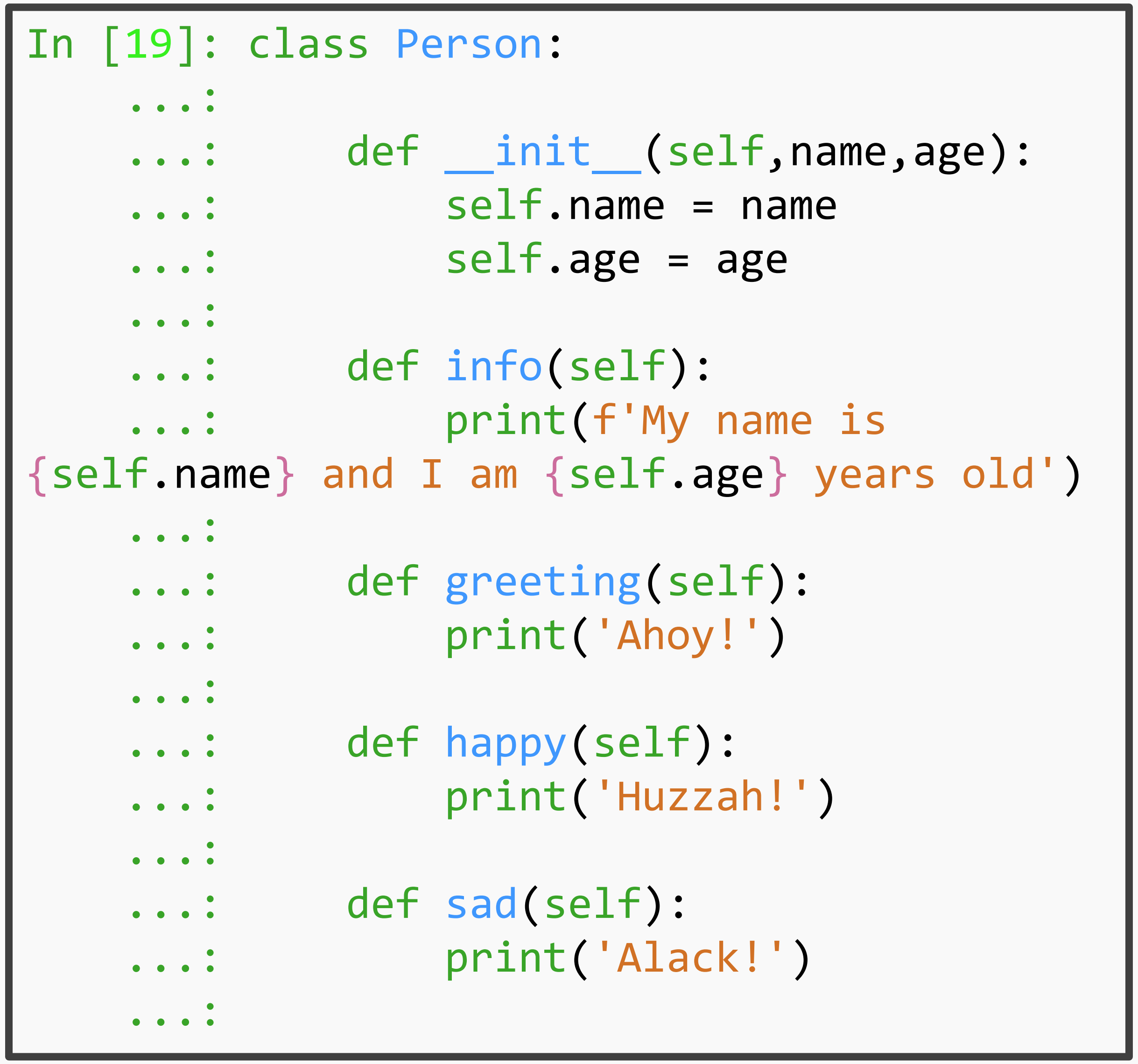
```
with open('filename.txt') as f:
   ...:     text = f.read()
```

⌛ Digestion Time


Strings
Lists
Dictionaries
Classes
Functions


Strings
Lists
Dictionaries
Classes
Functions

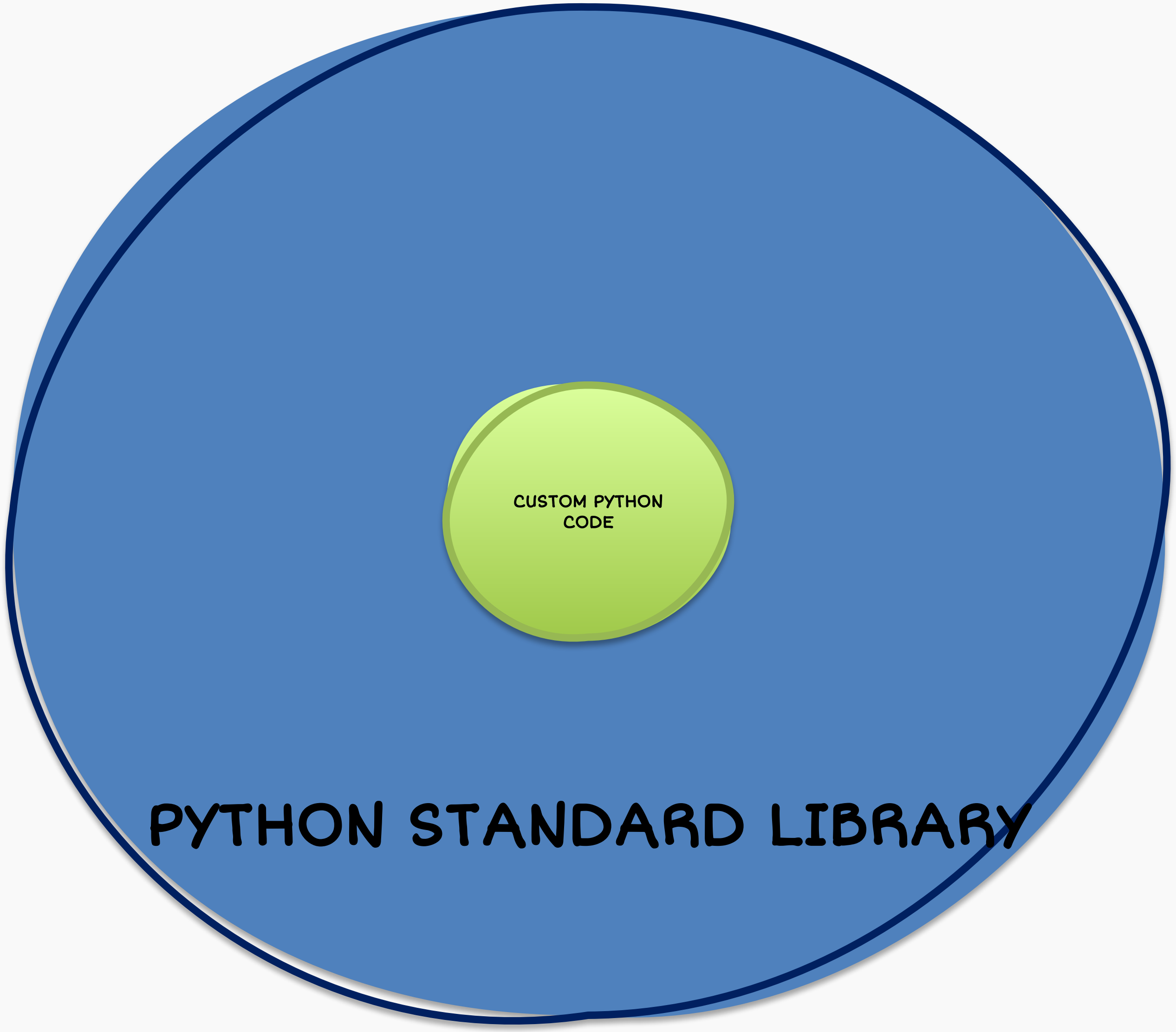
What makes Python special ?

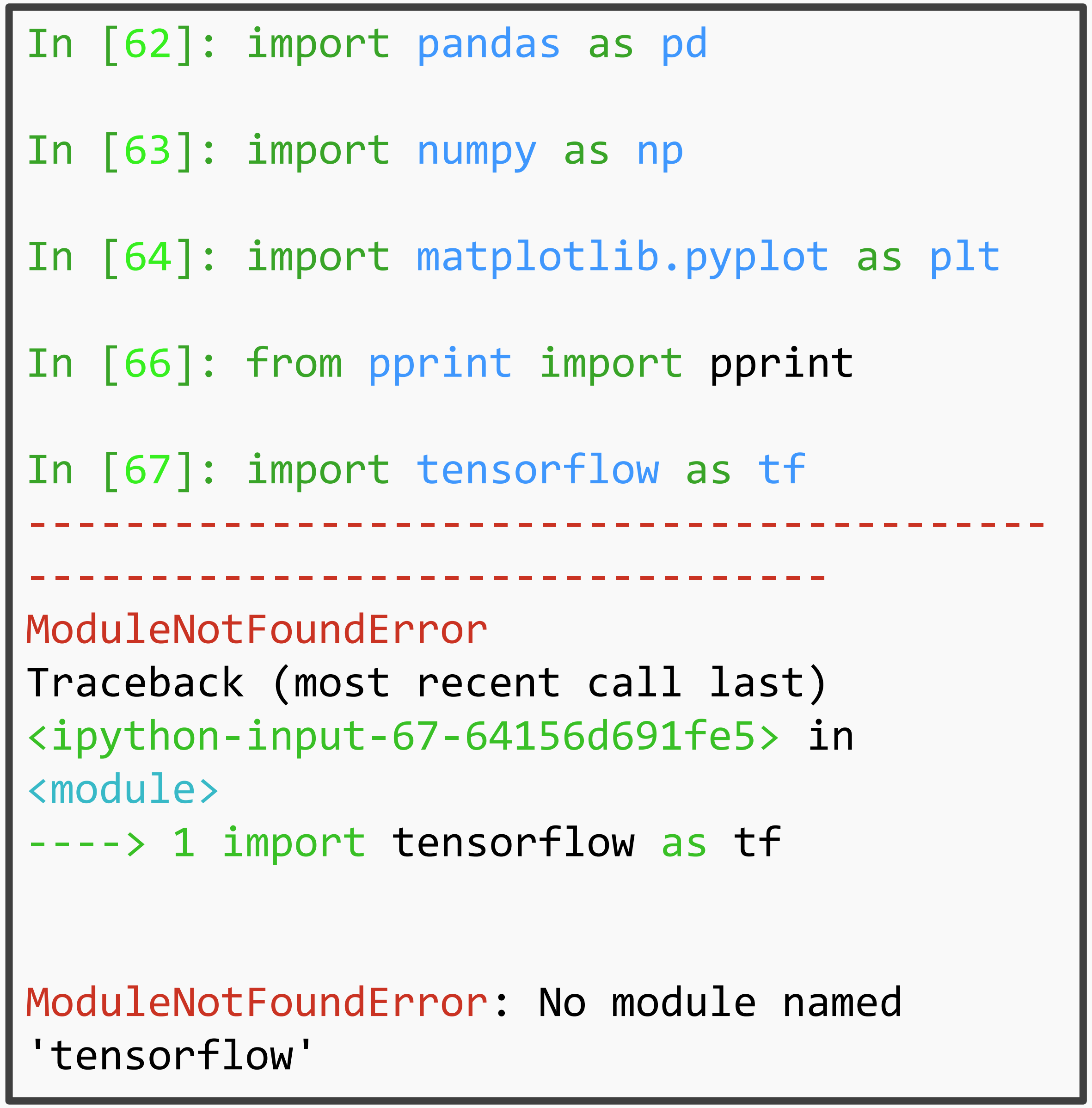


```
In [19]: class Person:
    ...:
    ...:     def __init__(self,name,age):
    ...:         self.name = name
    ...:         self.age = age
    ...:
    ...:     def info(self):
    ...:         print(f'My name is
{self.name} and I am {self.age} years old')
    ...:
    ...:     def greeting(self):
    ...:         print('Ahoy!')
    ...:
    ...:     def happy(self):
    ...:         print('Huzzah!')
    ...:
    ...:     def sad(self):
    ...:         print('Alack!')
    ...:
```

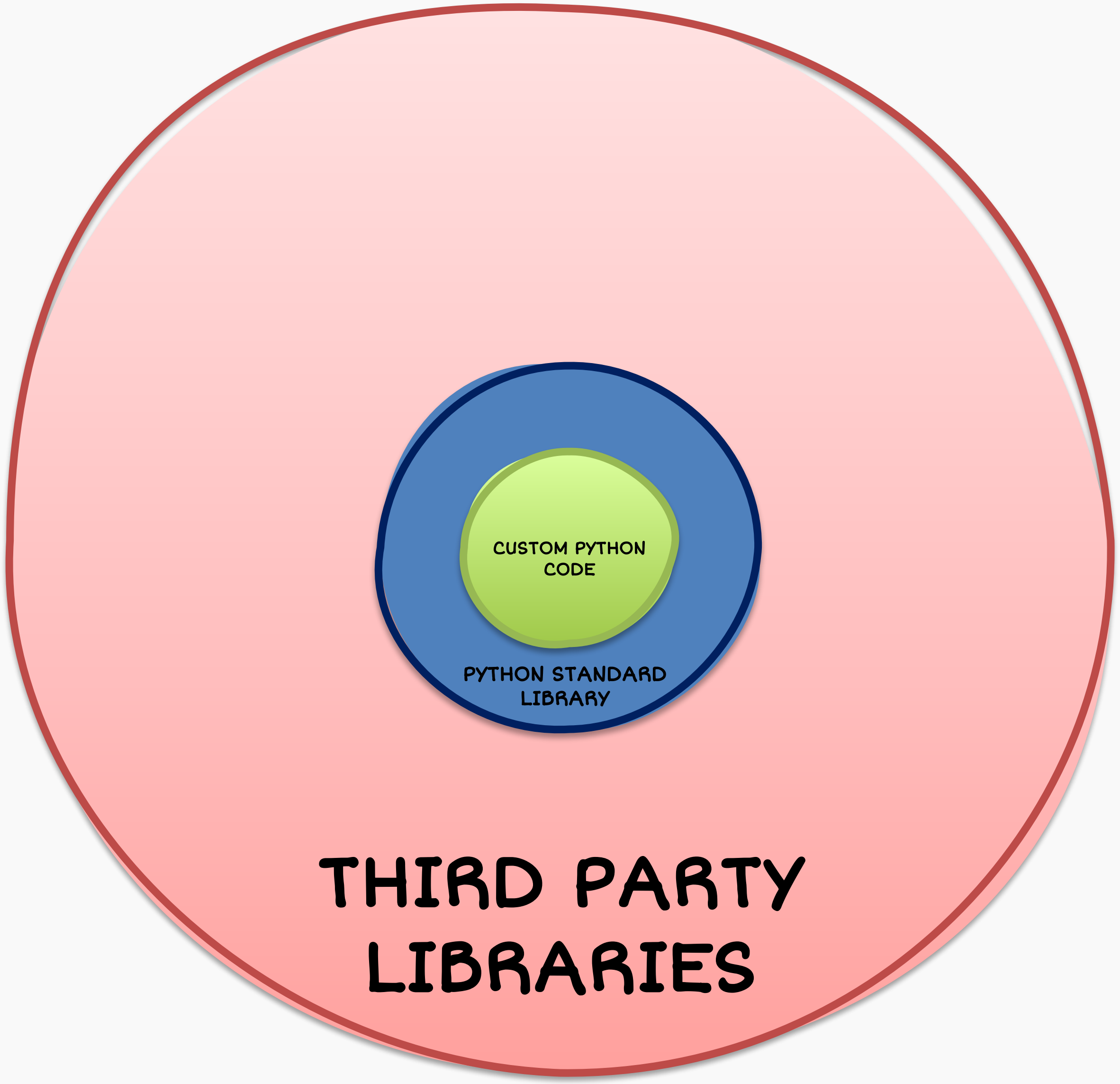
CUSTOM PYTHON CODE

```
In [34]: from time import sleep

In [35]: from random import randint

In [36]: from collections import defaultdict

In [37]: import traceback

In [38]: from sys import flags

In [39]: from datetime import datetime

In [40]: from functools import partial
```





```
In [62]: import pandas as pd

In [63]: import numpy as np

In [64]: import matplotlib.pyplot as plt

In [66]: from pprint import pprint

In [67]: import tensorflow as tf
-------------------------------------------
------------------------------------
ModuleNotFoundError
Traceback (most recent call last)
<ipython-input-67-64156d691fe5> in
<module>
----> 1 import tensorflow as tf


ModuleNotFoundError: No module named
'tensorflow'
```

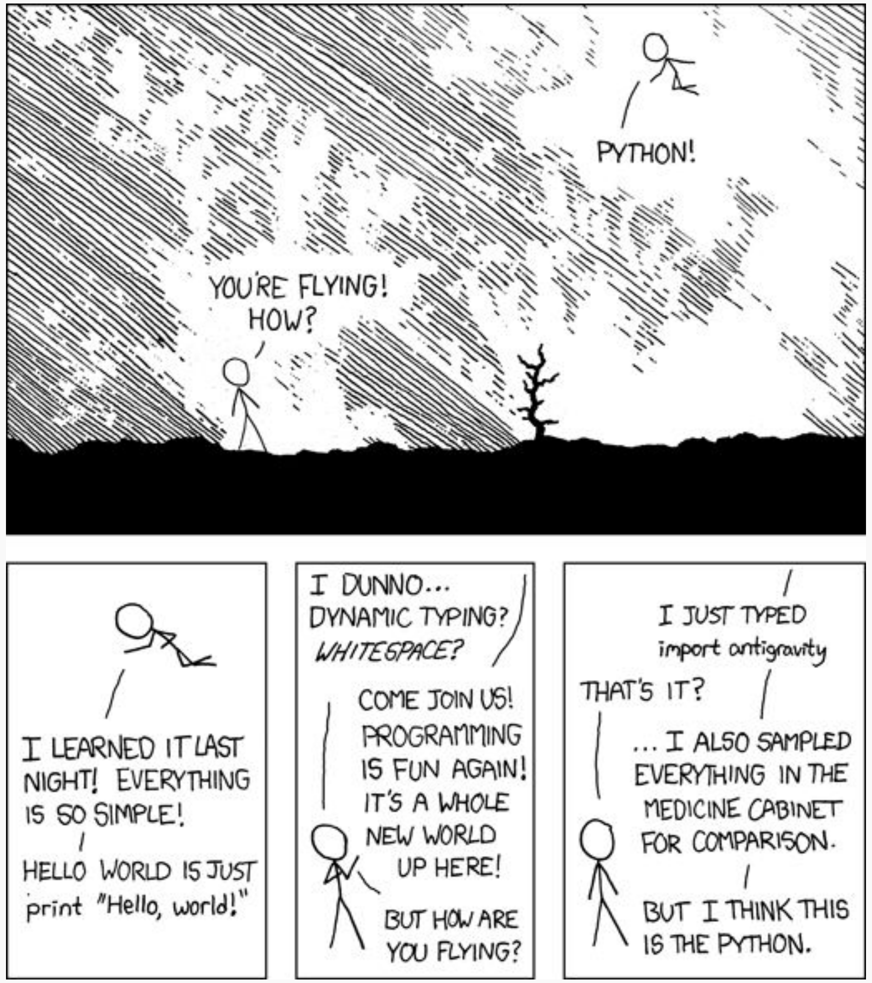


Python Libraries

- Python comes prepackaged with over 200 modules giving added functionality and ease of use.
- This collection is called the Standard library
- Along with this, Python supports thousands of third-party modules that can be downloaded using a package manager such as pip.

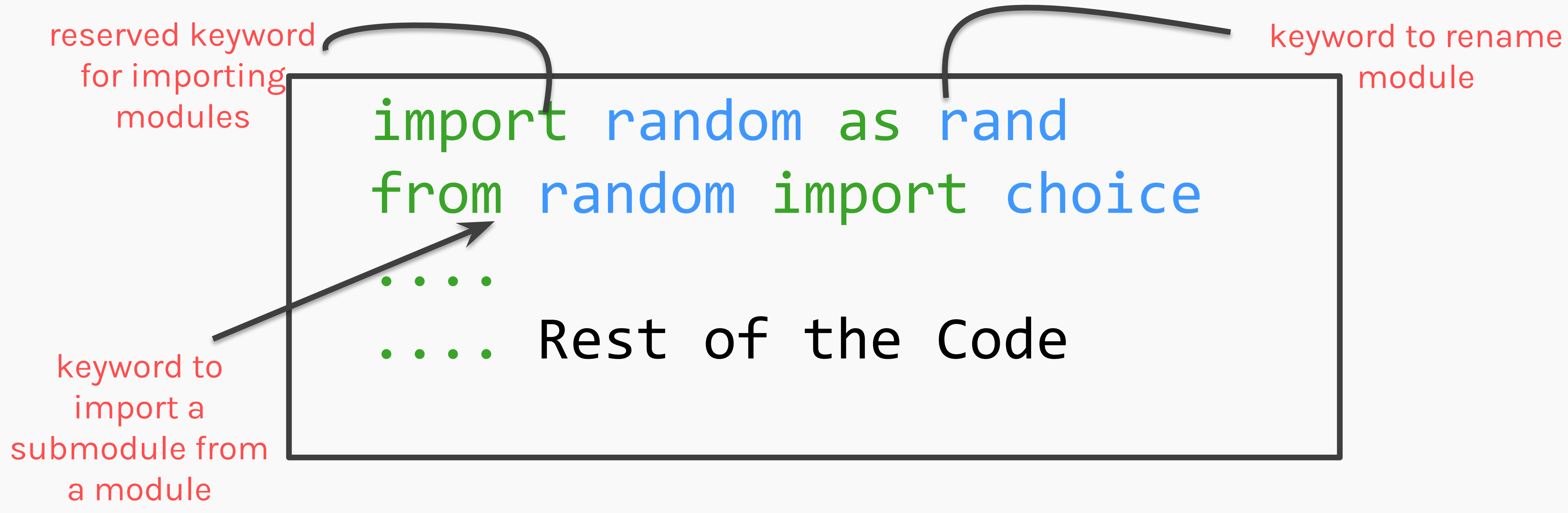
SYNTAX: `pip install library_name`

```
import antigravity
```


PYTHON!
YOU'RE FLYING! HOW?
I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"
I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?
I JUST TYPED import antigravity
THAT'S IT?
...I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.

Using the python library

You can use a library in your project by using the `import` statement


reserved keyword for importing modules
keyword to rename module
keyword to import a submodule from a module

```
import random as rand
from random import choice
....
.... Rest of the Code
```

⏳ Digestion Time

Regular Expressions

Regular expressions

- Regular expressions allow us to make generalized searches across text.
- This is possible through use of some pre-defined keywords which can be combined.
- They can also be used for find and replace operations in large swaths of text.

SYNTAX:`re.findall(pattern,mystr)`

```
import re
```

HOW TO REGEX
STEP 1: OPEN YOUR FAVORITE EDITOR
@GARABATOKID
STEP 2: LET YOUR CAT PLAY ON YOUR KEYBOARD
/^([A-Z0-9_\.-

Regular Expressions

Character Class	Represents	Example	Matches
.	Any character except a newline	'x.'	'x1', 'x ', 'x!', 'xZ', 'x😎', ...

Regular Expressions

Character Class	Represents	Example	Matches
.	Any character except a newline	'x.'	'x1', 'x ', 'x!', 'xZ', 'x😎', ...
\d	Any numeric digit from 0 to 9	'\d\d\d'	'528, '491', ...

Regular Expressions

Character Class	Represents	Example	Matches
.	Any character except a newline	'x.'	'x1', 'x ', 'x!', 'xZ', 'x😎', ...
\d	Any numeric digit from 0 to 9	'\d\d\d'	'528, '491', ...
\w	Any letter, digit, or an underscore (mnemonic: “word” characters)	'\w\w\w'	'Aar', '21B', '0_Z', ...

Regular Expressions

Character Class	Represents	Example	Matches
.	Any character except a newline	'x.'	'x1', 'x ', 'x!', 'xZ', 'x😎', ...
\d	Any numeric digit from 0 to 9	'\d\d\d'	'528, '491', ...
\w	Any letter, digit, or an underscore (mnemonic: "word" characters)	'\w\w\w'	'Aar', '21B', 'O_Z', ...
\s	Any space, tab, or newline character (mnemonic: "space" characters)	'\s\s'	' ', '\t\t', ' \n', ...

And a lot more ...

Regular Expressions

Character	Description	Example	Matches
?	Match zero or one of the preceding	"ab?"	"a", "ab", but not "abb"

Regular Expressions

Character	Description	Example	Matches
?	Match zero or one of the preceding	"ab?"	"a", "ab", but not "abb"
*	Match zero or more repetitions of preceding	"ab*"	"a", "ab", "abb", ...

Regular Expressions

Character	Description	Example	Matches
?	Match zero or one of the preceding	"ab?"	"a", "ab", but not "abb"
*	Match zero or more repetitions of preceding	"ab*"	"a", "ab", "abb", ...
+	Match one or more repetitions of preceding	"ab+"	"ab", "abb", "abbb"... but not "a"

Regular Expressions

Character	Description	Example	Matches
?	Match zero or one of the preceding	"ab?"	"a", "ab", but not "abb"
*	Match zero or more repetitions of preceding	"ab*"	"a", "ab", "abb", ...
+	Match one or more repetitions of preceding	"ab+"	"ab", "abb", "abbb"... but not "a"
{n}	Match n repetitions of preceding search	"ab{2}" "	"abb"

Regular Expressions

Character	Description	Example	Matches
?	Match zero or one of the preceding	"ab?"	"a", "ab", but not "abb"
*	Match zero or more repetitions of preceding	"ab*"	"a", "ab", "abb", ...
+	Match one or more repetitions of preceding	"ab+"	"ab", "abb", "abbb"... but not "a"
{n}	Match n repetitions of preceding search	"ab{2}" "	"abb"
{m,n}	Match between m and n repetitions of preceding	"ab{2,3}"	"abb" or "abbb"

Regular Expressions

BEFORE

```
In [98]: mystr = 'My phone number is 9920011914'
In [99]: pattern = '\d\d\d\d\d\d\d\d\d\d'
In [100]: re.findall(pattern,mystr)
Out[100]: ['9920011914']
```

AFTER

```
In [98]:mystr ='My phone number is 9920011914'
In [99]:pattern = '\d{10}'
In [100]:re.findall(pattern,mystr)
Out[100]:['9920011914']
```

Regular Expressions

Character	Description	Example	Matches
[ ]	Match a custom character set	"[xyz]"	"x", "y", and "z"

Regular Expressions

Character	Description	Example	Matches
[ ]	Match a custom character set	"[xyz]"	"x", "y", and "z"
[^]	Match the *complement* of a custom character set (i.e., *any* character *not* in the set)	"[^xyz]"	"$", "🦔", "7", "q", ..., but *not* "x", "y", or "z"

Regular Expressions

Character	Description	Example	Matches
[ ]	Match a custom character set	"[xyz]"	"x", "y", and "z"
[^]	Match the *complement* of a custom character set (i.e., *any* character <u>*not*</u> in the set)	"[^xyz]"	"$", "🦔", "7", "q", ..., but <u>*not*</u> "x", "y", or "z"
\W, \D, \S	Each matches the *complement* of their lowercase equivalent's character set	"\D\S"	"Z&", "@3", ..., but <u>*not*</u> "2X", "u ", ...

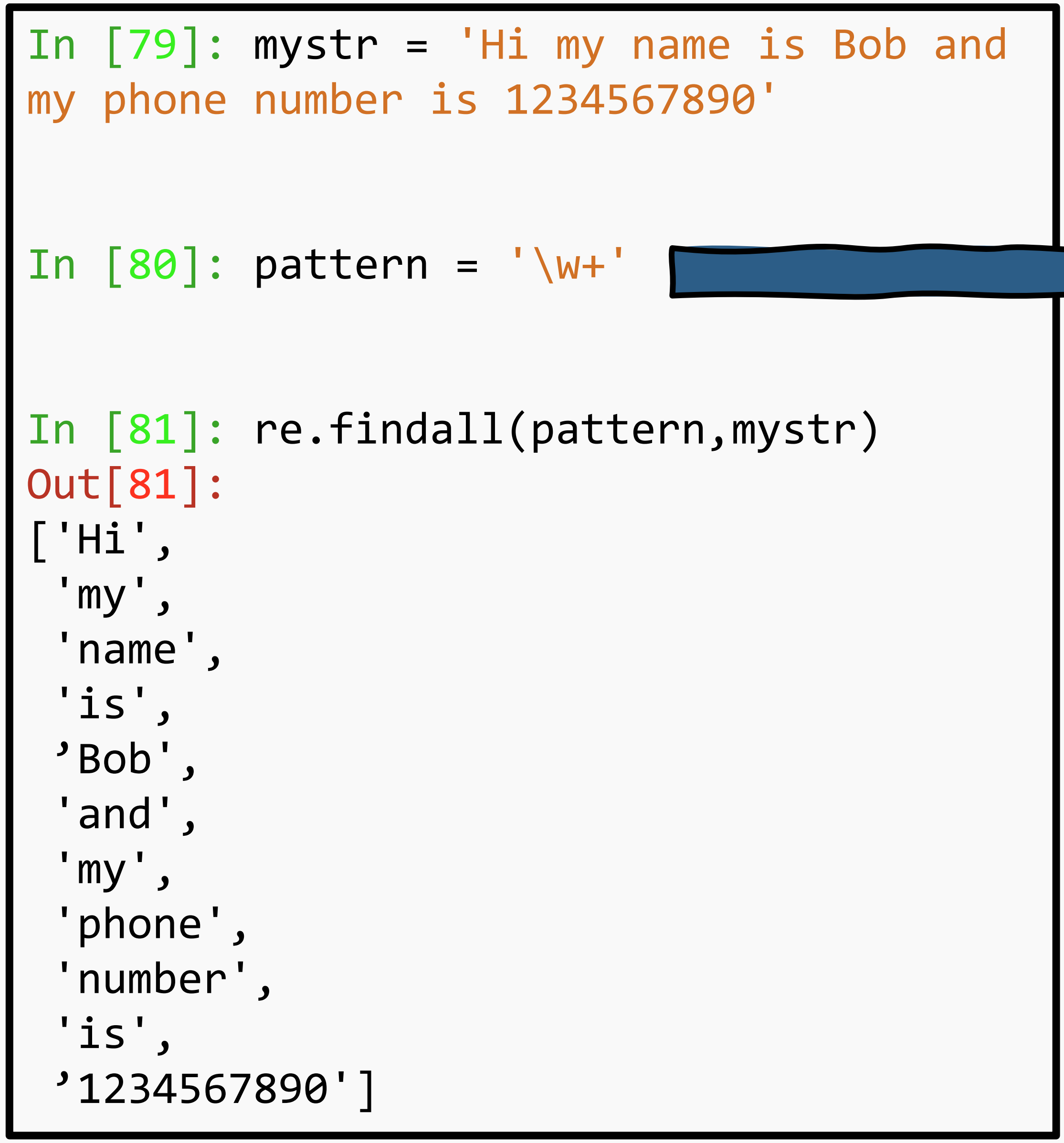
Regular Expressions

Character	**Description**	**Example**	**Matches**
[ ]	Match a custom character set	"[xyz]"	"x", "y", and "z"
[^]	Match the *complement* of a custom character set (i.e., *any* character *not* in the set)	"[^xyz]"	"$", "🦔", "7", "q", ..., but *not* "x", "y", or "z"
\W, \D, \S	Each matches the *complement* of their lowercase equivalent's character set	"\D\S"	"Z&", "@3", ..., but *not* "2X", "u ", ...
( )	Group substring into single token	"(ab)+"	"ab", "abab", "ababab", ...

Regular Expressions

Character	Description	Example	Matches
[ ]	Match a custom character set	"[xyz]"	"x", "y", and "z"
[^]	Match the *complement* of a custom character set (i.e., *any* character *not* in the set)	"[^xyz]"	"$", "🦔", "7", "q", ..., but *not* "x", "y", or "z"
\W, \D, \S	Each matches the *complement* of their lowercase equivalent's character set	"\D\S"	"Z&", "@3", ..., but *not* "2X", "u ", ...
( )	Group substring into single token	"(ab)+"	"ab", "abab", "ababab", ...
|	Logical 'or' (i.e., will match regex on either side)	"(e|o)utopia"	"eutopia", and "outopia"

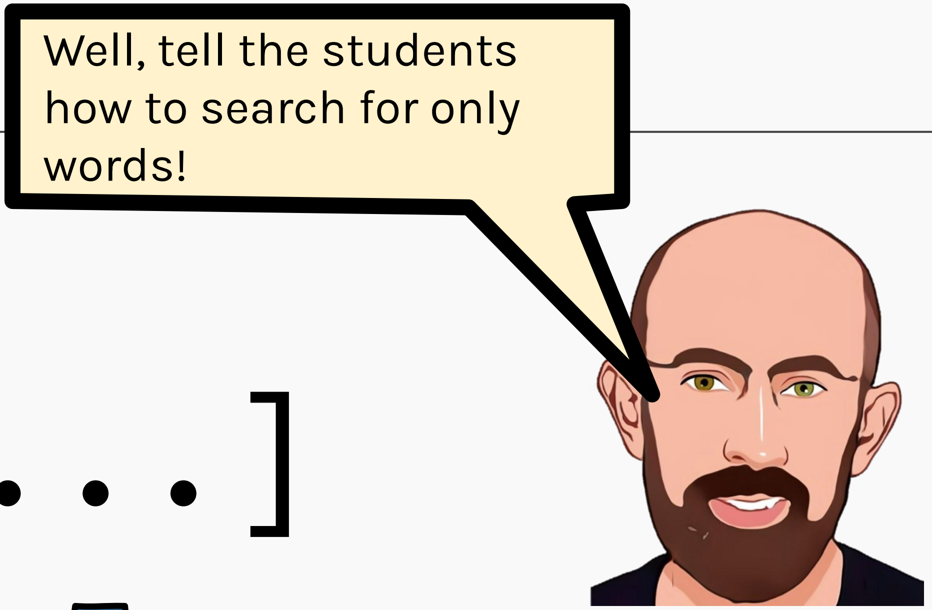
Custom character classes



```
In [79]: mystr = 'Hi my name is Bob and
my phone number is 1234567890'


In [80]: pattern = '\w+'


In [81]: re.findall(pattern,mystr)
Out[81]:
['Hi',
 'my',
 'name',
 'is',
 'Bob',
 'and',
 'my',
 'phone',
 'number',
 'is',
 '1234567890']
```

[...]

[A-Za-z]+



Regular Expressions

Character	Description	Example	Matches
^	Match start of line	"^Hello"	"Hello" but *not* "Oh, Hello"

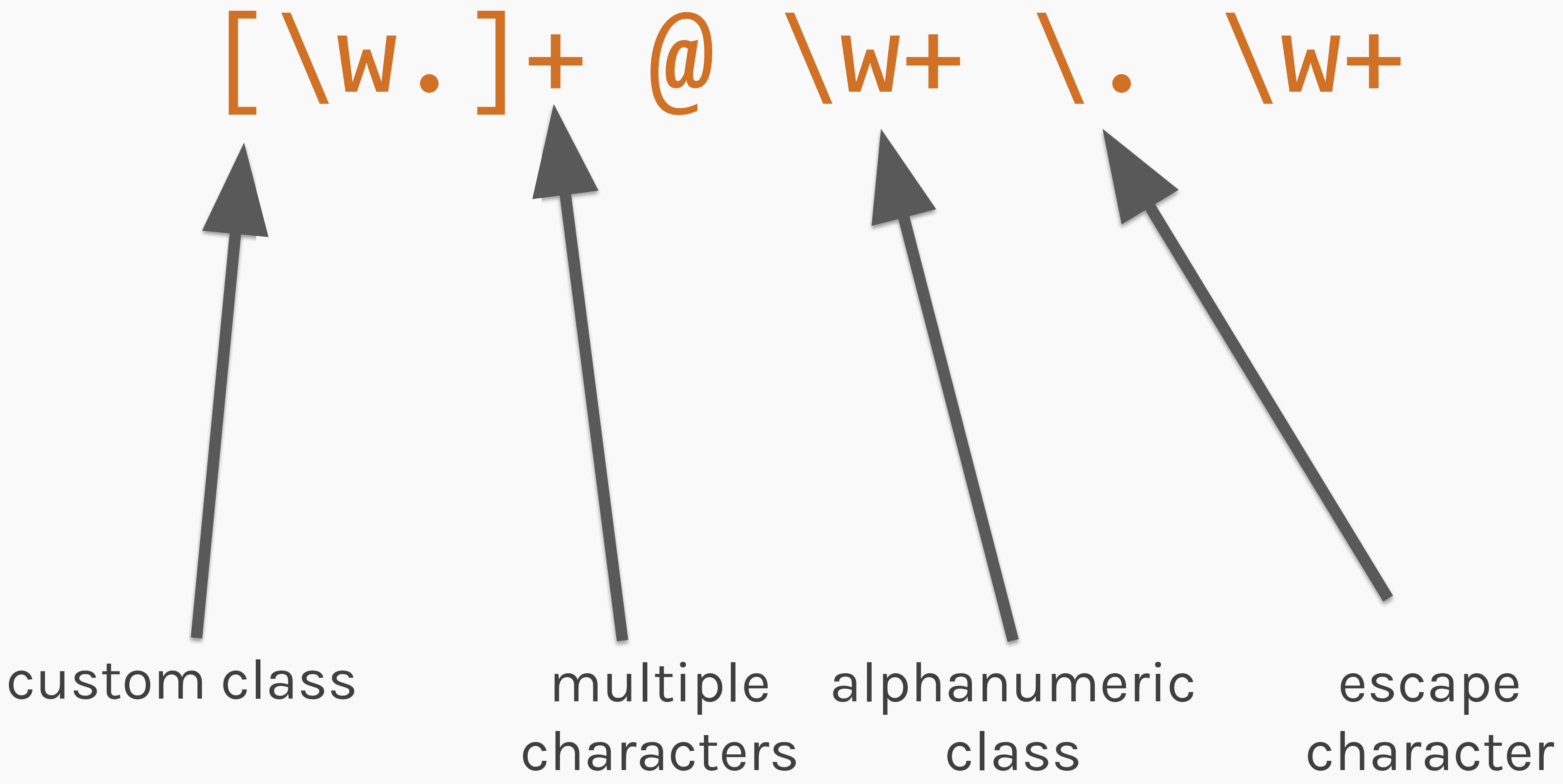
Regular Expressions

Character	Description	Example	Matches
^	Match start of line	"^Hello"	"Hello" but *not* "Oh, Hello"
$	Match end of line	"the end$"	"the end", but *not* "the end?"

Regular Expressions

Character	Description	Example	Matches
^	Match start of line	"^Hello"	"Hello" but *not* "Oh, Hello"
$	Match end of line	"the end$"	"the end", but *not* "the end?"
\	Escape special characters so they are interpreted literally (NOTE: behaves differently with certain sequences like "\w" or "\D")	"\$\.\^"	"$.^"

```
[\w.]+ @ \w+ \. \w+
```


custom class
multiple characters
alphanumeric class
escape character

Regular Expressions

```
import re
pattern = '[\w.]+@\w+\.\w+'
re.findall(pattern,'My email is bob@gmail.com')
>>>['bob@harvard.edu']
```

Regular Expressions

```
import re
pattern = '[\w.]+@\w+\.\w+'
re.findall(pattern,'My email is bob@gmail.com')
>>>['bob@harvard.edu']
```

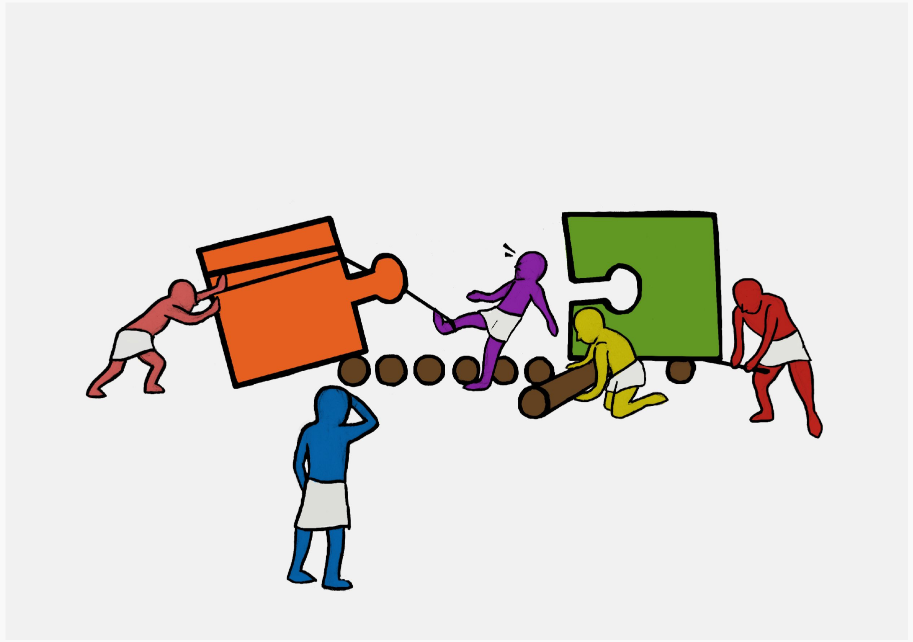
Regular Expressions

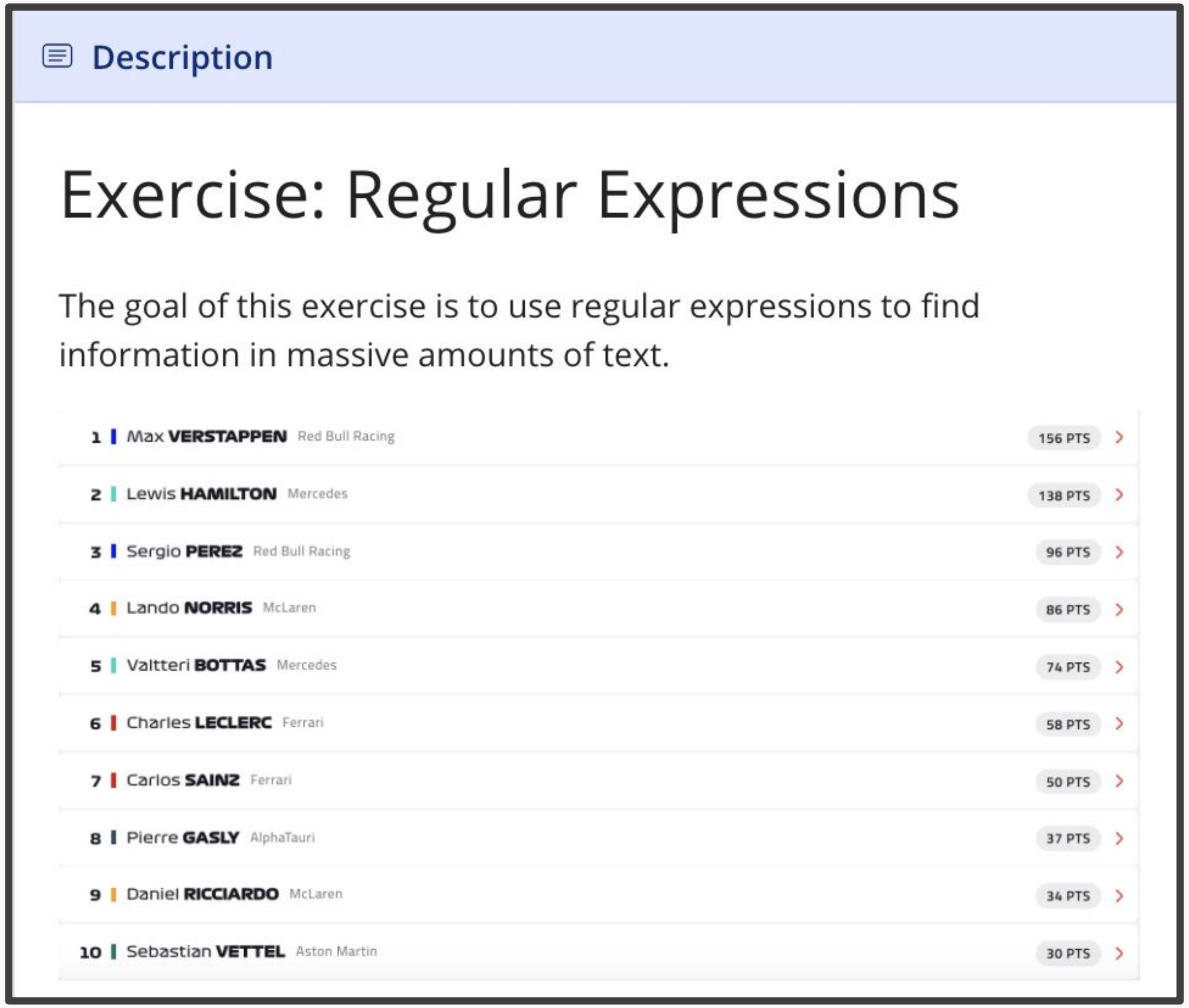
```
import re
pattern = '[\w.]+@\w+\.\w+'
re.findall(pattern,'My email is bob@gmail.com')
>>>['bob@harvard.edu']
```

Regular Expressions

```
import re
pattern = '[\w.]+@\w+\.\w+'
re.findall(pattern,'My email is bob@gmail.com')
>>>['bob@harvard.edu']
```

Exercise #2





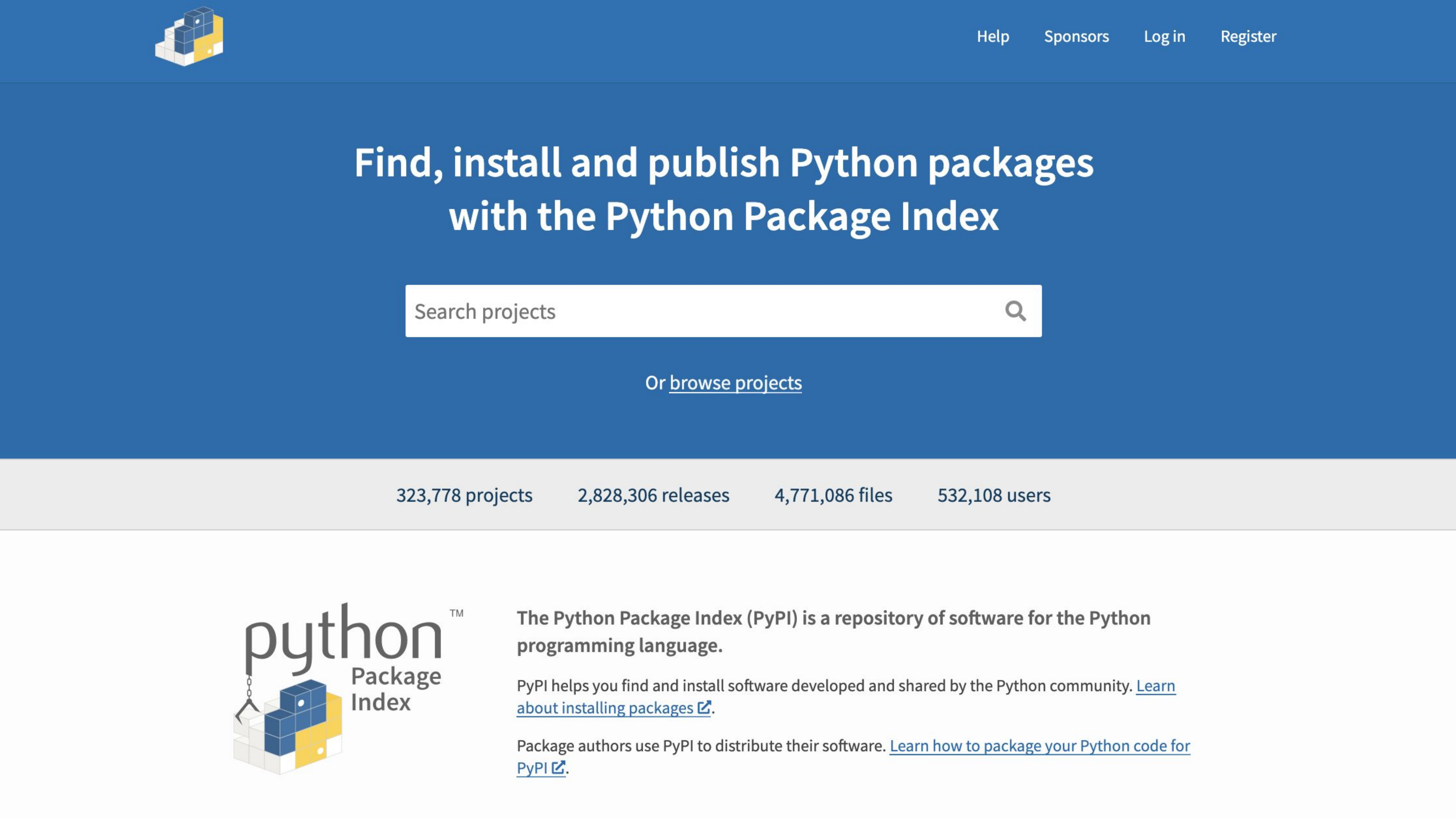
Description

Exercise: Regular Expressions

The goal of this exercise is to use regular expressions to find information in massive amounts of text.

1	Max **VERSTAPPEN**	Red Bull Racing	156 PTS
2	Lewis **HAMILTON**	Mercedes	138 PTS
3	Sergio **PEREZ**	Red Bull Racing	96 PTS
4	Lando **NORRIS**	McLaren	86 PTS
5	Valtteri **BOTTAS**	Mercedes	74 PTS
6	Charles **LECLERC**	Ferrari	58 PTS
7	Carlos **SAINZ**	Ferrari	50 PTS
8	Pierre **GASLY**	AlphaTauri	37 PTS
9	Daniel **RICCIARDO**	McLaren	34 PTS
10	Sebastian **VETTEL**	Aston Martin	30 PTS

Third party modules



Find, install and publish Python packages with the Python Package Index

Or browse projects

323,778 projects 2,828,306 releases 4,771,086 files 532,108 users

The Python Package Index (PyPI) is a repository of software for the Python programming language.

PyPI helps you find and install software developed and shared by the Python community. Learn about installing packages.

Package authors use PyPI to distribute their software. Learn how to package your Python code for PyPI.

File formats

.txt	Text Data
.csv	Tabular Data
.png	Image data
.mp4	Video data

Pandas

- Download pandas library using Python Package Index (PyPI)
- import pandas library (convenient to rename it)
- Use read_csv() function

```
!pip install pandas
import pandas as pd
df = pd.read_csv('filename.csv')
```