Day 2 - Lists - Dictionaries, & Tuples

Dodawanie, odejmowanie na listach nie działa co jest w miarę oczywiste. Raz, że sa tam i stringi i liczby a dwa, że są ustawione w określonych pozycjach. To łączenia, usuwania potrzebujemy wbudowanych funkcji

.pop()

.pop(3) #po pozycji

.append()

.append(jakas\_lista\_lub\_slownik\_lub\_cokolwiek) #zagnieżdżanie

.remove() #to istnieje jeszcze? Tak. Po nazwie usuwa

W słownikach też można zagnieżdżać listy, słowniki itd.

my\_dict[‘deep’] = my\_list #no wiesz o co chodzi

Day 4 - Conditionals - Expressions, if, else if (elif), else

mamma = „Bastek” #przypisanie

mamma == True #porównanie (czy zmienna mamma ma taką samą wartość jak True? Zwróci False)

Jeżeli w kodzie widzimy == oznacza to że sprawdzamy czy jeden element jest równy drugiemu to znaczy czy jest True lub False. I stosujemy ten mechanizm w naszych pętlach. Jeżeli coś jest True to zrób to i to. I po to są Boolean po to aby mówić programowi co ma robić w określonych warunkach tj. co się dzieje jeżeli coś jest True or False, on or off, yes or no…

my\_list = [1,2,3]

for i in my\_list:

if i == 2:

print(„yup it’s two”)

elif i == 1:

print(„it’s one man!”)

else:

print(„three”)

list\_d = [‘Applke’, ‘Mike’, 4324, ‘something’, ‘mummy’, 54325]

list\_e =[]

for i in list\_d:

if isinstance(i, int):

list\_e.append(i)

print list\_e

x = 0

list\_d = [‘Applke’, ‘Mike’, 4324, ‘something’, ‘mummy’, 54325]

list\_e =[]

for item in list\_d:

if isinstance(item, int):

list\_e.append(item)

list\_d.pop(x)

x += 1

Teraz mamy jedną listę z liczbami jedną ze stringami. Spójrz na x jak się zmienia i jak jest przekazywana wartość w pętli. Z każdym obrotem zwiększa się o jeden.

Day 5 – Functions

my\_list = [‘sf’, ‘Fdas’, ‘rE’, ‘rsDga’, ‘ds’, ‘rdag’]

my\_list.sort(key=str.lower, reverse=True)

odwrócone sortowanie. reverse=True jest nie potrzebne jak chcemy normalnie posortować.

new\_list = sorted(my\_list, key=str.lower, reverse=True)

jaka róznica? A no taka, że sorted nie zmienia zapisu w pamięcie kompytera listy my\_list a tylko mówi, hej niech zmienna new\_list będzie tym miejscem w pamięci tylko w sposób uporządkowany (ale samo my\_list pozostaje bez zmian)

items = ["Mic", "Phone", 323.12, 3123.123, "Justin", "Bag", "Cliff Bars", 134]

str\_items = []

num\_items = []

for i in items:

if isinstance(i, float) or isinstance(i, int):

num\_items.append(i)

elif isinstance(i, str):

str\_items.append(i)

else:

pass

print(str\_items)

print(num\_items)

def parse\_lists(abc):

str\_list\_items = []

num\_list\_items = []

for i in abc:

if isinstance(i, float) or isinstance(i, int):

num\_list\_items.append(i)

elif isinstance(i, str):

str\_list\_items.append(i)

else:

pass

return str\_list\_items, num\_list\_items

print(parse\_lists(items))

items3 = ["Mic", "Phone", 323.12, 3123.123, "Justin", "Bag", "Cliff Bars", 134]

#sum([123, 323, 423]) zwróć uwagę, że ta funkcje sumuje LISTĘ cyfr

def my\_sum(my\_num\_list):

total = 0

for i in my\_num\_list:

if isinstance(i, float) or isinstance(i, int):

total += i

return total

my\_sum(items3)

def count\_nums(my\_num\_list): #ile liczb w liście

total = 0

for i in my\_num\_list:

if isinstance(i, float) or isinstance(i, int):

total += 1

return total

def my\_avg(my\_num\_list): #średnia z liczb, które są w liście

the\_sum = my\_sum(my\_num\_list)

num\_of\_items = count\_nums(my\_num\_list)

return the\_sum / (num\_of\_items \* 1.0)

my\_avg(items3)

items3 = ["Mic", "Phone", 323.12, 3123.123, "Justin", "Bag", "Cliff Bars", 134]

def my\_sum\_and\_count(my\_num\_list):

total = 0

count = 0

for i in my\_num\_list:

if isinstance(i, float) or isinstance(i, int):

total += i

count += 1

return total, count

def my\_avg(my\_num\_list):

the\_sum, num\_of\_items = my\_sum\_and\_count(my\_num\_list)

return the\_sum / (num\_of\_items \* 1.0)

my\_avg(items3)

Day 6 - String Formatting, Substitution, and More Functions

https://github.com/codingforentrepreneurs/30-Days-of-Python/blob/master/PythonCheatSheet.md

powyższy link jest b.dobry. Obczajaj! Raus!

.split() nie tylko rozdziela w miejscu wskazanym ale przede wszystkim robi listę.

import datetime

default\_names = ["Justin", "john", "Emilee", "Jim", "Ron", "Sandra", "veronica", "Whitney"]

default\_amounts = [123.32, 94.23, 124.32, 323.4, 23, 322.122323, 32.4, 99.99]

unf\_message = """Hi {name}!

Thank you for the purchase on {date}.

We hope you are exicted about using it. Just as a

reminder the purcase total was ${total}.

Have a great one!

Team CFE

"""

def make\_messages(names, amounts):

messages = []

if len(names) == len(amounts):

i = 0

today = datetime.date.today()

text = '{today.month}/{today.day}/{today.year}'.format(today=today)

for name in names:

name = name[0].upper() + name[1:].lower()

new\_amount = "%.2f" %(amounts[i])

new\_msg = unf\_message.format(

name=name,

date=text,

total=new\_amount

)

i += 1

print(new\_msg)

make\_messages(default\_names, default\_amounts)

Ważne do zapamiętania. Komputer sczytuje z góry na dół i ZAPAMIĘTUJE! Spójrz na zmienną i. Wchodząc do pętli było 0 ale po kolejnym dodaniu zwiększa się o jeden. Co nie znaczy, że wychodzi z pętli. Zostaje tam tylko ciągle pamięta nową przypisaną wartość, która się zmienia z każdą pętlą. No i tu zastosowanie widać zmienności zmiennych 

Day 7 – Classes

class Animal():

name = 'Amy'

noise = "Grunt"

size = "Large"

color = "Brown"

hair = 'Covers body'

def get\_color(self):

return self.color

def make\_noise(self):

return self.noise

dog = Animal()

dog.make\_noise()

dog.size = "small"

dog.color = "black"

dog.hair = "hairless"

class Dog(Animal):

name = 'Jon'

size = "small"

color = "black"

age = 19

jon = Dog()

jon.color = 'white'

jon.name = 'Jon Snow'

Day 8 - Classes Part 2

class Animal():

name = 'Amy'

noise = "Grunt"

size = "Large"

color = "Brown"

hair = 'Covers body'

def get\_color(self, abc):

return self.color + " " + abc

@property

def make\_noise(self):

return self.noise

dog = Animal()

dog.get\_color("red")

dog.make\_noise

#arg = Positional Arguments

#kwarg = Keyword Argument

abc = "new string"

def some\_func(arg\_1, arg\_2, kwarg\_1=None, kwarg\_2=None):

print(arg\_1, arg\_2)

if kwarg\_1 != None:

print(kwarg\_1)

#return arg\_1

some\_func("abc", "car", kwarg\_1='Not a big deal')

email\_address = "another@gmail.com"

to\_list = ['abc@gmail.com']

from\_list = ['another2@gmail.com', 'hello@teamcfe.com']

def send\_email(email, to\_list=to\_list, from\_list=from\_list):

pass

send\_email("hello@teamcfe.com", to\_list=['abc@gmail.com'], from\_list=[['abc@gmail.com](mailto:'abc@gmail.com)'])

Day 9 - Classes Part 3, Importing, Custom Py Modules

import datetime

class MessageUser():

user\_details = []

messages = []

base\_message = """Hi {name}!

Thank you for the purchase on {date}.

We hope you are exicted about using it. Just as a

reminder the purcase total was ${total}.

Have a great one!

Team CFE

"""

def add\_user(self, name, amount, email=None):

name = name[0].upper() + name[1:].lower()

amount = "%.2f" %(amount)

detail = {

"name": name,

"amount": amount,

}

today = datetime.date.today()

date\_text = '{today.month}/{today.day}/{today.year}'.format(today=today)

detail['date'] = date\_text #dodajemy do słownika detail

if email is not None: # to samo co if email != None

detail["email"] = email

self.user\_details.append(detail) #dodajemy do listy

def get\_details(self):

return self.user\_details

def make\_messages(self):

if len(self.user\_details) > 0:

for detail in self.get\_details():

name = detail["name"]

amount = detail["amount"]

date = detail["date"]

message = self.base\_message

new\_msg = message.format(

name=name,

date=date,

total=amount

)

self.messages.append(new\_msg)

return self.messages

return []

Powyższe zapisujemy w pliku np. messages.py i używamy w terminalu poprzez

python messages.py

i na przykład:

obj = MessageUser()

obj.add\_user("Justin", 123.32, email='hello@teamcfe.com')

obj.add\_user("jOhn", 94.23)

obj.add\_user("Sean", 93.23)

obj.add\_user("Emilee", 193.23)

obj.add\_user("Marie", 13.23)

obj.get\_details()

obj.make\_messages()

a jeżeli byśmy mieli coś jeszcze w tym pliku, to znaczy jakąć definicję (funkcję) poza klasą to możemy

python messages.py make\_message

no tak jak w

python manage.py runserver

python manage.py strartapp

itd.

Generalnie w pliku .py zapisujemy klasy z funkcjami, mogą tam też być funkcje, zmienne, słowniki, cokolwiek. Tylko jeżeli wołamy w shell python to mysi to być formuła na najwyższym poziomie. Nie możemy na przykład wołać bezpośrednio funkcji która jest w klasie. Chyba tego nie możemy, bo Justin tego nie mówi. No ale co do zasady wołamy jak powyżej.

Kolejna rzecz, dostęp do pliku. Importowanie. Zasada jest prosta: ściezka dostępu zaczyna się na tym samym poziomie i kończy się na naszym pliku z rozszerzeniem .py.

from make\_message import get\_user

powyżej definicję którę importujemy ‘get\_user’ mamy w pliku make\_message.py, który jest na tym samym poziomie co nasz zapisywany plik (ten do którego inportujemy)

Jeżeli importujemy z głębi folderu, który jest na tym samym poziomie to przede wszystkim w tym folderze tworzymy plik \_\_init\_\_.py To znaczy ten plik \_\_init\_\_.py musi być na tym poziomie co plik z którego importujemy funkcje lub klasę. \_\_init\_\_.py mówi o tym, że ten folder jest folderem pythona i można go używać.

from folder.python\_plik import python\_def, python\_cośtam

from folder.inny\_plik import KlasaJakas, KlasaInna

from py\_day\_test import make\_messages

from py\_day\_mod.make\_messages import MessageUser

from random.whatever import antyhign

obj = MessageUser()

obj.add\_user("Abc", 123.32, email='hello@teamcfe.com')

obj.add\_user("jOhn", 94.23)

obj.add\_user("Sean", 93.23)

obj.add\_user("Emilee", 193.23)

obj.add\_user("Marie", 13.23)

obj.get\_details()

print(obj.make\_messages())

default\_names = ["Justin", "john", "Emilee", "Jim", "Ron", "Sandra", "veronica", "Whitney"]

default\_amounts = [123.32, 94.23, 124.32, 323.4, 23, 322.122323, 32.4, 99.99]

make\_messages(default\_names, default\_amounts)

Czyli lepiej sobie dzielić. W jednym miejscu zapisujemy nasze definicje i klasy a w drugim (tutaj) dane jakie mają przyjąć. I dopiero wtedy uruszamiamy shell python (maszyne wirtualną) i testujemy

python plik\_do\_wykonania\_z\_danymi

Day 10 - Setup Python to Send Email with Gmail

import smtplib

host = "smtp.gmail.com"

port = 587

username = "hungrypy@gmail.com"

password = "iamhungry2016"

from\_email = username

to\_list = ["hungrypy@gmail.com"]

email\_conn = smtplib.SMTP(host, port)

email\_conn.ehlo()

email\_conn.starttls()

email\_conn.login(username, password)

email\_conn.sendmail(from\_email, to\_list, "Hello there this is an email message")

email\_conn.quit()

from smtplib import SMTP

ABC = SMTP(host, port)

ABC.ehlo()

ABC.starttls()

ABC.login(username, password)

ABC.sendmail(from\_email, to\_list, "Hello there this is an email message")

ABC.quit()

from smtplib import SMTP, SMTPAuthenticationError, SMTPException

pass\_wrong = SMTP(host, port)

pass\_wrong.ehlo()

pass\_wrong.starttls()

try:

pass\_wrong.login(username, "wrong\_password")

pass\_wrong.sendmail(from\_email, to\_list, "Hello there this is an email message")

except SMTPAuthenticationError:

print("Could not login")

except:

print("an error occured")

pass\_wrong.quit()

https://docs.python.org/3/library/smtplib.html

# Day 11 - HTML & Plain Text Emails through Python and Gmail

tu brakuje pelnego repo na githubie ale dobra lekcja. Jak tworzyć html w wysyłanych wiadomościach. Aaaaaa jest w lekcji kolejnej. Spoko

<https://docs.python.org/3/library/email.mime.html>

i jeszcze o importowaniach. Jesli importujemy z biblioteki Pythona to tam szukamy jak zaimportować do pliku w powyższym linku importujemy na przykład:

from email.mime.base import MIMEBase

from email.mime.multipart import MIMEMultipart

# Day 12 - Send Formatted Emails to a Set of Users

import datetime

from email.mime.multipart import MIMEMultipart

from email.mime.text import MIMEText

import smtplib

host = "smtp.gmail.com"

port = 587

username = "hungrypy@gmail.com"

password = "iamhungry2016"

from\_email = username

to\_list = ["hungrypy@gmail.com"]

class MessageUser():

user\_details = []

messages = []

email\_messages = []

base\_message = """Hi {name}!

Thank you for the purchase on {date}.

We hope you are exicted about using it. Just as a

reminder the purcase total was ${total}.

Have a great one!

Team CFE

"""

def add\_user(self, name, amount, email=None):

name = name[0].upper() + name[1:].lower()

amount = "%.2f" %(amount)

detail = {

"name": name,

"amount": amount,

}

today = datetime.date.today()

date\_text = '{today.month}/{today.day}/{today.year}'.format(today=today)

detail['date'] = date\_text

if email is not None:

detail["email"] = email

self.user\_details.append(detail)

def get\_details(self):

return self.user\_details

def make\_messages(self):

if len(self.user\_details) > 0:

for detail in self.get\_details():

name = detail["name"]

amount = detail["amount"]

date = detail["date"]

message = self.base\_message

new\_msg = message.format(

name=name,

date=date,

total=amount

)

user\_email = detail.get("email")

if user\_email:

user\_data = {

"email": user\_email,

"message": new\_msg

}

self.email\_messages.append(user\_data)

else:

self.messages.append(new\_msg)

return self.messages

return []

def send\_email(self):

self.make\_messages()

if len(self.email\_messages) > 0:

for detail in self.email\_messages:

user\_email = detail['email']

user\_message = detail['message']

try:

email\_conn = smtplib.SMTP(host, port)

email\_conn.ehlo()

email\_conn.starttls()

email\_conn.login(username, password)

the\_msg = MIMEMultipart("alternative")

the\_msg['Subject'] = "Billing Update!"

the\_msg["From"] = from\_email

the\_msg["To"] = user\_email

part\_1 = MIMEText(user\_message, 'plain')

the\_msg.attach(part\_1)

email\_conn.sendmail(from\_email, [user\_email], the\_msg.as\_string())

email\_conn.quit()

except smtplib.SMTPException:

print("error sending message")

return True

return False

obj = MessageUser()

obj.add\_user("Justin", 123.32, email='hungrypy@gmail.com')

obj.add\_user("jOhn", 94.23, email='hungrypy@gmail.com')

obj.add\_user("Sean", 93.23, email='hungrypy@gmail.com')

obj.add\_user("Emilee", 193.23, email='hungrypy@gmail.com')

obj.add\_user("Marie", 13.23, email='hungrypy@gmail.com')

obj.get\_details()

obj.send\_email()

# Day 13 - Using External Template files with Context Data for Email Message

import os

def get\_template\_path(path):

file\_path = os.path.join(os.getcwd(), path)

if not os.path.isfile(file\_path):

raise Exception("This is not a valid template path %s"%(file\_path))

return file\_path

def get\_template(path):

file\_path = get\_template\_path(path)

return open(file\_path).read()

def render\_context(template\_string, context):

return template\_string.format(\*\*context)

file\_ = 'templates/email\_message.txt'

file\_html = 'templates/email\_message.html'

template = get\_template(file\_) #w otwartym pliku….

template\_html = get\_template(file\_html)

context = {

"name": "Justin",

"date": None,

"total": None

}

print(render\_context(template, context))

print(render\_context(template\_html, context))

# Day 14 - CSV Files with Python - Read, Write, & Append

import csv

with open("data.csv", "w+") as csvfile: #nadpisuje to co jest

writer = csv.writer(csvfile)

writer.writerow(["Col 1", "Col 2"])

writer.writerow(["Data 1", "Data 2"])

with open("data.csv", "r") as csvfile:

reader = csv.reader(csvfile)

for row in reader:

print(row)

with open("data.csv", "a") as csvfile: #dodaje na końcu tabeli

writer = csv.writer(csvfile)

writer.writerow(["Data 3", "Data 4"])

with open("data.csv", "r") as csvfile:

reader = csv.DictReader(csvfile)

for row in reader:

print(row)

with open("data.csv", "a") as csvfile:

fieldnames = ["id", "title"]

writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

writer.writeheader()

writer.writerow({"id": 123, "title": "New title"})

## Working with Files

```

file\_obj = open(file\_name, "<mode>")

```

| Mode | Description |

| ------------- | ------------- |

| "r" | Read only. Default mode. |

| "rb" | Read only in binary format |

| "r+" | Read and write |

| "rb+" | Read and write in binary format |

| "w" | Write only. Overwrites existing file or creates a new file. |

| "wb" | Write only in binary format. Overwrites existing file or creates a new file. |

| "w+" | Read and write. Overwrites existing file or creates a new file. |

| "wb+" | Read and write in binary format. Overwrites existing file or creates a new file. |

| "a" | Append to existing file or creates new file. |

| "ab" | Append to existing file or creates new file in binary format. |

| "a+" | Read and append. Overwrites existing file or creates a new file. |

| "ab+" | Read and append in binary format. Overwrites existing file or creates a new file. |

# Day 15 - Functions to Dynamically Add Data to CSV with Python

import csv

def get\_length(file\_path):

with open("data.csv") as csvfile:

reader = csv.reader(csvfile)

reader\_list = list(reader)

#print(reader\_list)

return len(reader\_list)

def append\_data(file\_path, name, email):

fieldnames = ['id', 'name', 'email']

#the number of rows?

next\_id = get\_length(file\_path)

with open(file\_path, "a") as csvfile:

writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

writer.writerow({

"id": next\_id,

"name": name,

"email": email,

})

append\_data("data.csv", "Justin", "hello@teamcfe.com")

# Day 16 - Edit CSV with Python

import csv

import datetime

import shutil

from tempfile import NamedTemporaryFile

def get\_length(file\_path):

with open("data.csv", "r") as csvfile:

reader = csv.reader(csvfile)

reader\_list = list(reader)

return len(reader\_list)

def append\_data(file\_path, name, email, amount):

fieldnames = ['id', 'name', 'email', 'amount', 'sent', 'date']

next\_id = get\_length(file\_path)

with open(file\_path, "a") as csvfile:

writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

writer.writerow({

"id": next\_id,

"name": name,

"email": email,

"sent": "",

"amount": amount,

"date": datetime.datetime.now()

})

def edit\_data(edit\_id=None, email=None, amount=None, sent=None):

filename = "data.csv"

temp\_file = NamedTemporaryFile(delete=False)

with open(filename, "rb") as csvfile, temp\_file:

reader = csv.DictReader(csvfile)

fieldnames = ['id', 'name', 'email', 'amount', 'sent', 'date']

writer = csv.DictWriter(temp\_file, fieldnames=fieldnames)

writer.writeheader()

for row in reader:

#print(row['id'] == 4)

if edit\_id is not None:

if int(row['id']) == int(edit\_id):

row['amount'] = amount

row['sent'] = sent

elif email is not None and edit\_id is None:

if str(row['email']) == str(email):

row['amount'] = amount

row['sent'] = sent

else:

pass

writer.writerow(row)

shutil.move(temp\_file.name, filename)

return True

return False

edit\_data(email='hello@teamcfe.com', amount=99.99, sent='')

po co jest ten plik tymczasowy? Nie do końca czaję :>

# Day 17 - Read Data Function for CSV File

import csv

#file\_item\_path = os.path.join(os.getcwd(), "data.csv")

file\_item\_path = os.path.join(os.path.dirname(\_\_file\_\_), "data.csv")

def read\_data(user\_id=None, email=None):

filename = file\_item\_path

with open(filename, "r") as csvfile:

reader = csv.DictReader(csvfile)

items = []

unknown\_user\_id = None

unknown\_email = None

for row in reader:

if user\_id is not None:

if int(user\_id) == int(row.get("id")):

return row

else:

unknown\_user\_id = user\_id

if email is not None:

if email == row.get("email"):

return row

else:

unknown\_email = email

if unknown\_user\_id is not None:

return "User id {user\_id} not found".format(user\_id=user\_id)

if unknown\_email is not None:

return "Email {email} not found".format(email=email)

return None

# Day 18 - Running Python Commands wih Arguments in Terminal

# A teraz historia o tym jak zrobić komendy których można uzywać w Terminalu. Na początku w folderze w którym mamy naszą funkcję (w tym przypadku jest to plik data\_manager.py z funkcją def read\_data tworzymy pliki \_\_init\_\_.py oraz \_\_main\_\_.py no i plik csv powinien być na tym samym poziomie. W pliku \_\_main\_\_.py tworzymy nasze argumenty, których będziemu uzywać w Terminalu:

# \_\_main\_\_.py

from argparse import ArgumentParser

from data\_manager import read\_data #importujemy funkcję

parser = ArgumentParser(prog="hungry") #w folderze ‘hungry’ to wszystko

parser.add\_argument("type", type=str, choices=['view', 'message'])

#parser.add\_argument("did\_send", type=str, choices=['true', 'false'])

parser.add\_argument('-id', '--user\_id', type=int)

parser.add\_argument('-e', '--email', type=str)

args = parser.parse\_args()

if args.type == "view":

print(read\_data(user\_id=args.user\_id))

print(read\_data(email=args.email))

elif args.type == "message":

print("send message")

# Day 19 - Integrating Part 1

# data\_class.py

import csv

import datetime

import shutil

import os

from tempfile import NamedTemporaryFile

from utils.templates import get\_template, render\_context

file\_item\_path = os.path.join(os.path.dirname(\_\_file\_\_), "data.csv")

class UserManager():

def message\_user(self):

file\_ = 'templates/email\_message.txt'

file\_html = 'templates/email\_message.html'

template = get\_template(file\_)

template\_html = get\_template(file\_html)

context = {

"name": "Justin",

"date": None,

"total": None

}

print(render\_context(template, context))

print(render\_context(template\_html, context))

return None

def get\_user\_data(self, user\_id=None, email=None):

filename = file\_item\_path

with open(filename, "r") as csvfile:

reader = csv.DictReader(csvfile)

items = []

unknown\_user\_id = None

unknown\_email = None

for row in reader:

if user\_id is not None:

if int(user\_id) == int(row.get("id")):

return row

else:

unknown\_user\_id = user\_id

if email is not None:

if email == row.get("email"):

return row

else:

unknown\_email = email

if unknown\_user\_id is not None:

return "User id {user\_id} not found".format(user\_id=user\_id)

if unknown\_email is not None:

return "Email {email} not found".format(email=email)

return None

\_\_main\_\_.py

from argparse import ArgumentParser

from data\_class import UserManager

from utils.templates import get\_template, render\_context

parser = ArgumentParser(prog="hungry")

parser.add\_argument("type", type=str, choices=['view', 'message'])

parser.add\_argument('-id', '--user\_id', type=int)

parser.add\_argument('-e', '--email', type=str)

args = parser.parse\_args()

if args.type == "view":

print(UserManager().get\_user\_data(user\_id=args.user\_id, email=args.email))

elif args.type == "message":

print(UserManager().message\_user())

# Day 20 - Integrating Part 2

import csv

import datetime

from email.mime.multipart import MIMEMultipart

from email.mime.text import MIMEText

import smtplib

import shutil

import os

from tempfile import NamedTemporaryFile

from utils.templates import get\_template, render\_context

file\_item\_path = os.path.join(os.path.dirname(\_\_file\_\_), "data.csv")

host = "smtp.gmail.com"

port = 587

username = "hungrypy@gmail.com"

password = "iamhungry2016day19"

from\_email = username

to\_list = ["hungrypy@gmail.com"]

class UserManager():

def render\_message(self, user\_data):

file\_ = 'templates/email\_message.txt'

file\_html = 'templates/email\_message.html'

template = get\_template(file\_)

template\_html = get\_template(file\_html)

if isinstance(user\_data, dict):

context = user\_data

plain\_ = render\_context(template, context)

html\_ = render\_context(template\_html, context)

return (plain\_, html\_)

return (None, None)

def message\_user(self, user\_id=None, email=None, subject="Billing Update!"):

user = self.get\_user\_data(user\_id=user\_id, email=email)

if user:

plain\_, html\_ = self.render\_message(user)

print(plain\_, html\_)

user\_email = user.get("email", "hello@teamcfe.com")

to\_list.append(user\_email)

try:

email\_conn = smtplib.SMTP(host, port)

email\_conn.ehlo()

email\_conn.starttls()

email\_conn.login(username, password)

the\_msg = MIMEMultipart("alternative")

the\_msg['Subject'] = subject

the\_msg["From"] = from\_email

the\_msg["To"] = user\_email

part\_1 = MIMEText(plain\_, 'plain')

part\_2 = MIMEText(html\_, "html")

the\_msg.attach(part\_1)

the\_msg.attach(part\_2)

email\_conn.sendmail(from\_email, to\_list, the\_msg.as\_string())

email\_conn.quit()

except smtplib.SMTPException:

print("error sending message")

return None

def get\_user\_data(self, user\_id=None, email=None):

filename = file\_item\_path

with open(filename, "r") as csvfile:

reader = csv.DictReader(csvfile)

items = []

unknown\_user\_id = None

unknown\_email = None

for row in reader:

if user\_id is not None:

if int(user\_id) == int(row.get("id")):

return row

else:

unknown\_user\_id = user\_id

if email is not None:

if email == row.get("email"):

return row

else:

unknown\_email = email

if unknown\_user\_id is not None:

print("User id {user\_id} not found".format(user\_id=user\_id))

if unknown\_email is not None:

print("Email {email} not found".format(email=email))

return None

# Day 21-23 - Web Scraping with Python 3 Python Requests & BeautifulSoup

import requests

from bs4 import BeautifulSoup

base\_url = 'https://www.yelp.com/search?find\_desc=Restaurants&find\_loc='

loc = 'Newport+Beach,+CA'

page = 10

url = base\_url + loc + "&start=" + str(page)

yelp\_r = requests.get(url)

yelp\_soup = BeautifulSoup(yelp\_r.text, 'html.parser')

businesses = yelp\_soup.findAll('div', {'class': 'biz-listing-large'})

for biz in businesses:

title = biz.findAll('a', {'class': 'biz-name'})[0].text

print(title)

address = biz.findAll('address')[0] #.replace(' ', '')

print(address)

print('\n')

phone = biz.findAll('span', {'class': 'biz-phone'})[0].text

print(phone)

file\_path = 'yelp-{loc}.txt'.format(loc=loc)

with open(file\_path, "a") as textfile:

businesses = yelp\_soup.findAll('div', {'class': 'biz-listing-large'})

for biz in businesses:

title = biz.findAll('a', {'class': 'biz-name'})[0].text

print(title)

address = biz.findAll('address')[0] #.replace(' ', '')

print(address)

print('\n')

phone = biz.findAll('span', {'class': 'biz-phone'})[0].text

print(phone)

page\_line = "{title}\n{address}\n{phone}\n\n".format(

title=title,

address=address,

phone = phone

)

textfile.write(page\_line)

print(yelp\_soup.findAll('li', {'class': 'regular-search-result'}))

print(yelp\_soup.findAll('a', {'class': 'biz-name'}))

for name in yelp\_soup.findAll('a', {'class': 'biz-name'}):

print(name.text)

print(yelp\_r.status\_code) #200

print(yelp\_soup.prettify())

print(yelp\_soup.findAll('a', {}))

for link in yelp\_soup.findAll('a'):

print(link)

# Day 24 - Web Scraping Part 4

import requests

from bs4 import BeautifulSoup

base\_url = 'https://www.yelp.com/search?find\_desc=Restaurants&find\_loc='

loc = 'Newport+Beach,+CA,+United+States'

current\_page = 0

while current\_page < 201:

print(current\_page) #w sumie nie potrzebne

url = base\_url + loc + "&start=" + str(current\_page)

yelp\_r = requests.get(url)

yelp\_soup = BeautifulSoup(yelp\_r.text, 'html.parser')

businesses = yelp\_soup.findAll('div', {'class': 'biz-listing-large'})

file\_path = 'yelp-{loc}-2.txt'.format(loc=loc)

with open(file\_path, "a") as textfile:

businesses = yelp\_soup.findAll('div', {'class': 'biz-listing-large'})

for biz in businesses:

title = biz.findAll('a', {'class': 'biz-name'})[0].text

print(title)

second\_line = ""

first\_line = ""

try:

address = biz.findAll('address')[0].contents

for item in address:

if "br" in str(item):

second\_line += item.getText().strip(" \n\t\r")

else:

first\_line = item.strip(" \n\t\r")

print(first\_line)

print(second\_line)

except:

pass

print('\n')

try:

phone = biz.findAll('span', {'class': 'biz-phone'})[0].getText().strip(" \n\t\r")

except:

phone = None

print(phone)

page\_line = "{title}\n{address\_1}\n{address\_2}\n{phone}\n\n".format(

title=title,

address\_1=first\_line,

address\_2=second\_line,

phone = phone

)

textfile.write(page\_line)

current\_page += 10

no więc generalnie to patrz w url, jak się on zmienia i w html, jak on wygląda. Od tego zależy wszystko potem w kodzie.

I to o czym pisałem poprzednio. Przeciez to wsztsrko możesz zamknąć w definicję/klasę i używać w terminalu. Trzeba by tylko stworzyć definicję z atrybutem localisation gdzie będzie się wpisywać miejscowość.

# Day 25 - Web Scraping on Javascript-Driven HTML using Python

# Okay, a co jeśli strona jest dynamiczna i chcemy z niej obrazki które się zmieniają np. Courusel. Wtedy jest zupełnie inna historia. Samo Requests za bardzo nam nie pomoże. Generalnie kiedy słuszymy o scraping ze stronki czegoś co cechuję się uzyciem JS, jQuery też, czyli, jeśli to co chcemy wyciągnąć nosi znamiona jakiejś dynamiki to musimy użyć SELENIUM

# http://selenium-python.readthedocs.io/installation.html

# pip install selenium

# Te selenium to taka nakładka na FireFoxa która z nim współpracjuje w pobieraniu plików dynamicznych z przeglądarki. Taka dodatkowa obudowa dla FireFoxa. To nie znaczy, że nie potrzebujemy requests! Request służy nam do **wypakowania** tych plików, BeautyfullSoup do ich obsługi.

# Jest jeszcze cos takiego:

# https://pypi.python.org/pypi/ImageScraper

# ale tutaj i tak używamy też selenium, bo selenium jest tylko po to aby można było dynamicznie zczytać dane ze strony.

import os

import shutil

import time

import requests

from bs4 import BeautifulSoup

from selenium import webdriver

url = "http://www.chrisburkard.com/"

web\_r = requests.get(url)

web\_soup = BeautifulSoup(web\_r.text, 'html.parser')

print(web\_soup.findAll("img"))

driver = webdriver.Firefox()

driver.get(url)

iterations = 0

while iterations < 10:

html = driver.execute\_script("return document.documentElement.outerHTML")

sel\_soup = BeautifulSoup(html, 'html.parser')

images = []

for i in sel\_soup.findAll("img"):

src = i["src"]

images.append(src)

print(images)

current\_path = os.getcwd()

for img in images:

try:

file\_name = os.path.basename(img)

img\_r = requests.get(img, stream=True)

new\_path = os.path.join(current\_path, "images", file\_name)

with open(new\_path, "wb") as output\_file:

shutil.copyfileobj(img\_r.raw, output\_file)

del img\_r

except:

pass

iterations += 1

time.sleep(5)

# Day 26 - Get Data with an API

import json

import requests

from requests\_oauthlib import OAuth1

consumer\_key= 'NZTJrWhij8kemtAXmfyhyA'

consumer\_secret = 'JtcAesDkKuNltcKdwR7NEaUkgm8'

token = 'TCipoxU\_lYo55-F3rS10XdnN6f-3-KQI'

token\_secret = '5gAZ99Arn2x\_LIOVM25AWy8H84c'

url ='https://api.yelp.com/v2/search?term=food&location=San+Francisco'

r = requests.get(url, auth=auth)

def do\_search(term='Food', location='San Francisco'):

base\_url = 'https://api.yelp.com/v2/search'

term = term.replace(' ', '+')

location = location.replace(' ', '+')

url = "{base\_url}?term={term}&location={location}".format(base\_url=base\_url,

term=term,

location=location)

auth = OAuth1(consumer\_key,

consumer\_secret,

token,

token\_secret)

r = requests.get(url, auth=auth)

return r.json(), r.text

json\_data, text\_data = do\_search()

python\_data = json.loads(text\_data)

print(json.dumps(json\_data, indent=4, sort\_keys=True))

for i in json\_data['businesses']:

print(i["name"])

print(i["phone"])

print(i["location"]["display\_address"])

print(i["location"]["city"])