Python - Crash Course - 3rd edi

by

ERIC MATTHES

- No Starch Press

ISBN-13 : 978-1-7185-0270-3 (PRIN

Pages : 514

2. Variables & Simple Data Types

23/02/2024

(2)

variable inside a String ⇒ f-string =>

f"{variable1} is {variable2}"

e.g.: f" Name is {full_name.title()}"
↳ Title case

whitespace: \n & \t allowed inside a string

Removing whitespace:

string.rstrip() - ~~left side~~ - right side
.lstrip() - ~~right~~ - left side
.strip() - ~~only~~ both the sides.

? How to remove all whitespaces in a string?

string.replace(occurance, replacement)
(e.g.) string.replace(" ", "")

Removing prefixes:

string.removeprefix(prefixString)

? How to remove suffix? .removesuffix(~~s~~suffixstring)

? How to remove pattern occurrence anywhere?
string.replace(occurance, replacement);

All integer divides will result in float output

underscores allowed in numbers:
e.g.: 14_00_000

multiple assignments:

x, y, z = 10, 56, -5

Comment ⇒ #

>>> import this
⇒ Shows python programming principle

26/02/2024

3. Introducing Lists

(2)

[element1, element2, ----, elementn]

⇒ mixed data type

index position starts at 0

last item index = -1

last but one = -2 and so on

appending element ⇒)

list.append(element)

Empty list = []

inserting element ⇒ list.insert(index, element)

removing element ⇒ del <element>

remove & use element ⇒ list.pop()
at last index

remove at any position ⇒ list.pop(index)

remove by value ⇒ list.remove(value)

permanent sort ⇒ list.sort() ⇒ alphabetical

~~list.sort()~~ ⇒ reverse alphabetical

list.sort(reverse=True)

temporary sort ⇒ sorted(list)

~~permanent~~ reverse order ⇒ ~~sorted~~ list.reverse()
~~printing in~~ (not sorting)

Length of list ⇒ len(list)

27/02/2024

4. Working with LISTS

3.5

for loop ⇒

```
for item in <list>:
    <indent><
            body.
        >
```

~~for~~ range() ⇒

```
for value in range(1,5):
    <
        >
```

start with → 1, end before 5

↳ 1, 2, 3, 4 (not 5)

~~for value in range(1,5):~~

mylist = list(range(1,5))

↳ convert to list.

even_numbers = list(range(2,11,2))

↳ Step Value

Statistics ⇒ min(<list>)
max(<list>)
sum(<list>)

(*) List Comprehensions ⇒ (w/o using for loop).

squares = [value**2 for value in range(1,11)]

output: list of squares

Slicing a List ⇒ mylist[0:3] ⇒ element at index 0, 1, 2
mylist[:3] ⇒ from 0th element
mylist[2:] ⇒ from index-2 to end.
(**) mylist[-3:] ⇒ last 3 elements.
mylist[:] ⇒ entire list

copying a list ⇒

new_list = mylist[:]

↳ a new copy of a list

yourlist = mylist

↳ points to same list (mylist here)

How to know by address or object? → id()

Tuple ⇒ Immutable List (used for fixed/constant values)

Similar to List but with parantheses - instead of square bracket

(e.g.) matrix_dim = (12, 12) ⇒ 12 x 12 std dimension

⇒ Entire tuple can be redefined

(e.g.): my-dim = (12, 12)

my-dim = (10, 9)

5. IF Statements

```
if <condition>:
    <      >
elif <condition>:
    <      >
else:
    <      >
```

optional

== equal
!= not equal.
and ⇒ <cond1> and <cond2>
or = <cond1> or <cond2>

Value in a list?

value in <list>

Value not in a list?

value not in <list>

Boolean ⇒ True
False

Empty list?

```
if <list>:
    <not empty action>
else:
    <empty action>
```

6. Dictionaries

28/02/2024

(3)

- key-value pair

dict = { <key> : <value>, ... }

(e.g.): alien_0 = {'color' : 'green', 'points' : 5}

so, points = alien_0['points']

* Adding new pair ⇒

alien_0['x_pos'] = 0

alien_0['y_pos'] = 25

* Empty dict ⇒ alien_0 = {}

Modifying value ⇒ alien_0['color'] = 'yellow'

Removing a pair ⇒ del alien_0['points']

(*) Read a value with exception handling (e.g: no such key)

(e.g.) ~~alien~~

point_value = alien_0.get('points', 'No such value')

↓

No such key in this dict.

(e.g):

point_value = alien_0.get('points'):

∴ here it returns special value None

Looping:

① for key, value in <dict>.items():

    body

② for key in <dict>.keys(): ⇒ only keys

Looping (contd.)

(3) for key in sorted(<dict>.keys()): => keys in sorted order.

(4) for value in <dict>.values(): => only values in dict.

(*) Sets: Braces, but no key-value pair

(e.g): languages = {'python', 'rust', 'c'}

Nesting - List of Dictionaries:

~~list~~ mylist = [<dict1>, <dict2>, ... <dictn>]

Nesting - List in a Dictionary:

(e.g:)
```
pizza = {
    'crust' : 'thick',
    'toppings' : ['mushrooms', 'extra-cheese']
}
```

Nesting - Dictionary in a Dictionary

(e.g.):
```
users = {
    'swamy' : {
        'first_name' : 'Karappuswamy',
        'last_name' : 'Thanyaraj',
    },
    'Sangeethe' : {
        'firstname' : 'Sangeethe',
        'last_name' : 'Alagavel',
    },
}
```

7. User input & while loops 29/02/2024

input_value = input ("What is your name?") (1)

input a number => All inputs are strings. So use int() to convert to number.

modulo operator => % => returns remainder of two numbers divided.

4 % 3 => 1

While loops:

while <condition>:

Condition examples:

number <= 5

status => boolean

~~for~~

Exit a loop => break

Continue to next iteration skipping rest of body => continue

Loop with ~~list~~ List =>

while <list>: → each iteration fetches a value till it completes.

<value = list.pop()

:

:

>

Loop with a specific item in a list =>

while <item> in <list>:

01/03/2024

1.5

8. Functions

```
def <function-name>(<args>):
    <
        body
    >
```

- Arguments ⇒ Let def describe_pet(type, name):
  <
  >

Positional args
- order of args

(e.g.): describe_pet('dog', 'whillie')

Keyword args.
- name-value pairs in any order

(e.g.):
describe_pet(name='aillie'
type='dog')

- Default values ⇒

```
def describe_pet(name, type='dog'):
    <
    >
```

default arg should be last one

describe_pet(name='willie')
(or)
describe_pet('willie')

- Return values ⇒

return <simple value> (or) <dict> (or) <list>

- Optional argument ⇒

def get_name(first_name, last_name, middle_name='')

Passing a list ⇒

def greet_users(<username_list>):

modifying a list in a function ⇒

(*). The modifications are permanent.

(*) Preventing a function from Modifying a List ⇒
(Sending a copy)

Call ⇒ functionname(list[:])

↳ copy of list.

(*) Passing an arbitrary number of arguments ⇒

(1) def function(*arg):

↳ arg. of Tuple

(e.g) (1) ('pepperoni')

(2) ('mushroom', 'pepperoni', 'cheese')

(2) ~~def function(*arg, <regular positional arg>)~~

Mixing with regular positional argument ⇒

def function(size, *toppings):

(*) Using arbitrary keyword arguments ⇒

- Not knowing ahead of time what kind of information will be passed to the function.

(e.g) def build_profile(first, last, **user_info):

```
    return user_info
```

user_profile = build_profile('albert', 'einstein',
location='princeton',
field='physics')

02/03/2024

(Sat) 6

Modules => a file ending in .py

(e.g) pizza.py

```
def make_pizza(...)
    <
    >
```

make_pizza.py

```
import pizza
pizza.make_pizza(...)
```

=> importing specific functions only

(e.g)1: from pizza import make_pizza

2: from pizza import make_pizza,
deliver_pizza,
make_bill ...

=> Alias for a function

(e.g): from pizza import make_pizza as mkpz

=> Alias for a module

(e.g) import pizza as pz

=> importing all functions in a module

(e.g) ~~import~~ from pizza import *

9. CLASSES

OOP

```
class <classname>:
    def __init__(self, <args>):
        < self.arg1 = arg1
        > self.argn = argn

    def method1(self):
        <
        >
```

```
my_instance = <classname>(<args>)
~~my_method = f~~
my_instance.method1()
```

setting default value of attribute => in __init__(),
self.<attribute> = <value>

Inheritance => child class

```
class <child class name>(<parent class name>):
    def __init__(self, <args>):
        super().__init__(<args>)

my_child_instance = <child class name>(<args>)
```

overriding method from the parent class =>

```
def <method name same as parent's>(self):
    <
    >
```

If overridden method has different parameters than parent?
— still it will override

Instances as Attributes =>
create an instance of another class wi
constructor of new class.

Importing multiple classes from a module =
→ put put all classes in a module
~~→ import *~~
~~for~~

→ ~~from~~ <module> import *
from <module> import <specific clas
~~from import~~

10. FILES & EXCEPTIONS

Reading from a file:

```
from pathlib import Path
path = Path(<filename>)
```

↳ either absolute or relative path with forward slash (/)

```
contents = path.read_text()
```
=> Entire file

```
lines = contents.splitlines()
```
=> format it as list of lines.

```
for line in lines:
    print(line)
```

```
if <pattern> in contents:
    <
    >
```
=> check for a pattern in file content

Writing to a file:

```
path.write_text(<string content>)
```

↳ only writes string, numeric data to be converted to string.

↳ seperated by \n for newlines.

Exceptions =>

```
try:
    < code which may throw exception > => print(5/0)
except <exception object>: => ZeroDivisionError
    <exception handler> => print("Divide by zero is illegal")
else:
    <normal handler>
```

= Try exception throwing start with try-except block.

FileNotFoundError exception =>

```
try:
    contents = path.read_text(<filename>
except FileNotFoundError:
    print(contents)
    print("unable to read file")
else:
    print(contents)
```

Note: content.split() returns ~~no. of words~~ list of words.

- multiple files handling =>

```
filenames = ['file1', 'file2', ... 'filen']
for filename in filenames:
    path = Path(filename)
```

- failing silently =>

```
except <exception object>:
    pass
```

Storing Data ⇒

JSON – JavaScript Object Notation

~~from json~~

```
import json
path1 = Path(<jsonfile>)
contents1 = json.dumps(<data>)
```

(e.g.) list

⇒ writing data as json format

```
path1.write_text(contents1)

path2 = Path(<jsonfile>)
contents2 = path2.read_text()
<data> = json.loads(contents2)
```

(e.g.) list

- Note: check if file exists ⇒

```
if path.exists():
    <
    >
```

03/03/2024

11. TESTING YOUR CODE (2)

Installing [pytest] =>

~~$ python -m pip~~

$ python -m pip install --upgrade

$ python -m pip install --upgrade pip ↵

$ python -m pip install --user pytest ↵

Testing a Function =>

(1) my-functionfile.py

```
def myfunction( ):
    <
    >
```

(2) Now need to write a pytest file as below:

test_my-functionfile.py (mandatory)

```
from myfunctionfile import
                        myfunction
def test_<testname>():
    <"""call the myfunction to be
                        tested.
    assert <condition>
    >
```

(3) Run test: $ pytest ↵

04/03/2024

Testing a Class =>

(1)

Using Fixtures => helps to setup a test environment

```
import pytest
from <tested module> import <test class>

@pytest.fixture
def <common function-for all tests>():
    <
    return <common_data>
    >
```

→ input for all other tests follow

```
def test_test1(common_data):
    <
    >

def test_test2(common_data):
    <
    >
```

05/03/2024

④

Non-Gaming

Learnings from PROJECTS

PROJECT-1

12. A Ship that Fires Bullets

① sys.exit() → to exit python application

② if __name__ == '__main__':

⇒ this check is to test whether this python file is the top level one called from shell.

③ If double underscore is missing in __init__ of constructor (even if single underscore), python doesn't throw error, it silently ignores and considers it as just another function. So this constructor function won't be called during object creation.

Page 232

④ It is convention that helper function name is ~~pet r~~ prefix with a underscore (_).

⑤ float(value) returns floating point value when you want to preserve decimals.

⑥ Added feature in game:
While shooting a bullet, a wav file is played for nice game effect.

13. ALIENS

06/03/2024

(4)

Adding delay in seconds ⇒

~~ima~~ from time import sleep

sleep(<secs>)

⊗ ~~Added~~ Alien-Hit & Ship-Hit audio effects.

14. SCORING

08/03/2024

(3)

Pg: 281

10/03/2024

(2)

formatted string ⇒

f"{<variable/Data>:,}

↳ add , in appropriate decimal places.

PROJECT-2

11/03/2024

(3)

15. GENERATING DATA

matplotlib

```
import matplotlib.Pyplot as plt
```

(1)

Single plot ← squares = [1, 4, 9, 16, 25]

Collection of plots ← fig, ax = plt.subplots()

ax.plot(squares) → optional: linewidth

plt.show()

X is missing
- Must enter x params for meaningful plot

ax.set_title( )
ax.set_xlabel( )
ax.set_ylabel( )
→ optional fontsize

ax.tick_params( )

~~plt.style~~

plt.style.use( )

(2)

~~ax.sab~~

ax.scatter(<x>, <y>) → to plot single point

optional: color = < >

→ optional: s = <n> thickness of dot

+ single value or list of value

```
x_values = range(1, 1001)
y_values = [x**2 for x in x_values]
ax.axis([0, 1100, 0, 1100000])
```

x range, y range

plt.savefig( ) → to save plot as figure

Random walks

```
from random import choice
```

choice(<list of values>) ⇒
- returns a random value from the list.
- this is used to generate x & y values for data set used to plot.

Rolling Dice with Plotly

— Install plotly & pandas —

```
from random import randint
```

randint (<From value>, <To value>)
↳ returns in given range

Here die rolled 100 times & results are analysed - and found no. of times each face (side) is known
⇒ frequencies[ ], poss_results = [1, 2, 3, 4, 5, 6]

Histogram for above Die rolling case:

```
import plotly.express as px

fig = px.bar(x=poss_results,
             y=frequencies)
fig.show()
```

px.bar() optionals ⇒ title = "___"
labels = {'x': <x label>, 'y': <y label>}

fig.write_html(<HTML file name.html>)

16. DOWNLOADING DATA

12|03|2024

(2)

CSV File handling :

```
from pathlib import Path
import csv

path = Path(<path of file>)
lines = path.read_text().splitlines()

reader = csv.reader(lines)        <- an object of a row
header_row = next(reader)
print(header_row)

for index, column_header in
                enumerate(header_row):
    print(index, column_header)   <- column details extracted

for row in reader:
    <here row[0] - 1st column of row
     row[4] - 5th column of row>
```

date time handling =>

```
from datetime import datetime

date = datetime.strptime(
        '2021-07-01', '%Y-%m-%d')   <- date object
```

Error handling => (missing data on csv)

```
try:
    <extract column value>
except ValueError:
    print(<index> of the column)
else:
    <normal processing>
```

14/03/2024

2:5

JSON File handling

~~from~~ ~~import~~ pathlib import Path
import json

path = Path("<file path .json")
contents = path.read_text()

dictionary <= json_data = json.loads(contents)

len(json_data) => No of records.

< iterate through dictionary key and prepare list of data >

building World map =>

import plotly.express as px

~~fig~~ = px.scatter_go(lati, long, title)
fig.show()

optional:

size => ~~Size of~~ Size of dot

color_continuous_scale = "colors"
labels = {'color': 'magnitude'}
projections = natural earth

17. WORKING WITH APIS

$ python -m pip install --user requests

```
import requests
r = requests.get (<url with query>,
                  <headers>)
r.status_code => status code.
response_dict = r.json()
```

* Aware of API Rate limit
* Most probably you need API key or access token.

15/03/2024

PROJECT-3

18. Getting Started with DJANGO (2)

↓ Web framework

- create venv

```
$ python -m venv ll_env
$ source ll_env/bin/activate
(venv)$ deactivate
$ pip install --upgrade pip
$ pip install django
$ django-dr $ ls
$ django-admin startproject <projectname> .   ll_project
$ ls
$ python manage.py migrate
$ ls
$ python manage.py runserver <port no>   top terminal.
```

making Pages

16/03/2024 (1)

making Pages (contd.)

17/03/2024 (3)

19. USER ACCOUNTS

18/03/2024

(2)

forms => from django import forms

setting up Users Accounts

19/03/2024

(3)

SQLITE3

Additional learning while debugging:

apt install sqlite3 ↵

(11_env)$ python manage.py dbshell ↵

↑ Sqlite commands.

)

20. STYLING AND DEPLOYING AN APP

20/03/2024

(1)

Styling an app for various devices =>

django-bootstrap5 app

21/03/2024

(2)

Bug fixing done:

1. Topics - navbar link right aligned due to improper usage.
2. 'Edit Entry' added a new entry instead of editing due to wrong html called.

— End —