



```
[ ] print(f'{name} is a {job} and {age} years old.')
```

Ternary operator

```
[ ] y = 30  
x = 10 if y == 30 else 20  
print(x)
```

```
[ ] y = 30  
x = 10 if (y == 10) else 20 if (y == -10) else 40  
print(x)
```

```
def high_order_func(func, num_list):  
    return func(func(num_list))  
print(high_order_func(sum_of_nums, [1, 2, 3, 4, 5]))  
print(high_order_func(power_of_nums, [1, 2, 3, 4, 5]))
```

Generator (on the fly iteration)

```
[10] def num_generator(num):  
    while True:  
        yield num  
        num += 1  
plus_one = num_generator(3)  
print(next(plus_one)) # 3  
print(next(plus_one)) # 4  
# start
```

PYTHON TIPS AND TRICKS

HANDY RECIPES FOR FASTER AND BETTER CODING

Resul CALISKAN



Who is this book for

For those who want short summary information to level up in less time and become a master of python without reading a pile of books.





The content of this book

- ▶ # Templating strings
- ▶ # Concatenating
- ▶ # Splitting
- ▶ # Reversing
- ▶ # Printing out multiples of strings
- ▶ # Removing unnecessary characters on strings
- ▶ # Anagram Strings
- ▶ # Palindrome Strings
- ▶ # Frequency of elements in a List - Counter()
- ▶ # Using itertools (combinations)
- ▶ # Unique elements in a string
- ▶ # Swapping values
- ▶ # Returning multiple values from a function



The content of this book

- ▶ # Ternary operator
- ▶ # One line conditional if-else
- ▶ # Chained comparison
- ▶ # Chained function call
- ▶ # Slicing a list
- ▶ # Unpacking list
- ▶ # Using *args & **kwargs arguments
- ▶ # Copying List
- ▶ # List of list flattening
- ▶ # Min and max index in list
- ▶ # Using List Comprehensions
- ▶ # Using Dict&Set Comprehensions



The content of this book

- ▶ # Merging Dictionaries
- ▶ # Iterating over a dictionary
- ▶ # Inverting dictionary
- ▶ # Sorting dictionary by value
- ▶ # Most frequent element in a list or string.
- ▶ # Dictionary get method
- ▶ # Memory usage of an object.
- ▶ # Path of imported modules
- ▶ # getpass module
- ▶ # Using enumerate to get index/value pairs
- ▶ # Zip(*iterables) Function
- ▶ # Zip(*iterables) Function



The content of this book

- ▶ # Recursive function
- ▶ # Calculate the execution time
- ▶ # Files, directories of the current working directory.
- ▶ # Apply a function to every item in an iterable- `map(function, iterable)`
- ▶ # Lambda expressions-One line function.
- ▶ # Sorting with lambda function
- ▶ # Reduce function -`reduce(function, list)`
- ▶ # Filter function-`filter(function, list)`
- ▶ # Open two files
- ▶ # To query a request service
- ▶ # Higher order functions
- ▶ # Generator (on the fly iteration)
- ▶ # Using coroutine and decorators
- ▶ # Error handling "try-except-else-finally" blocks



For friends who want to master or advance in Python, I have compiled and rewrote most of the python code patterns available in Literature. Of course, if you want to be a good python coding master, not only read these codes, but also you need to practice these code patterns.



Let's not forget that the code patterns here are prepared to give you talent. You can use these code patterns exactly in your work and of course it is always possible to develop newer practical methods. Each project has its own unique requirements. For this reason, it is very important to write code in programming logic by taking these project requirements into account while writing code for your projects.

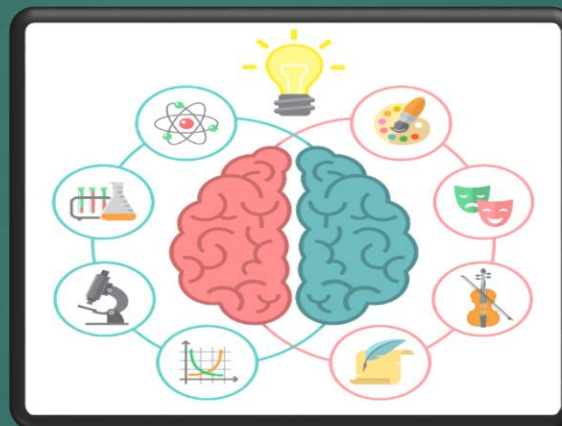


Skill Requirements

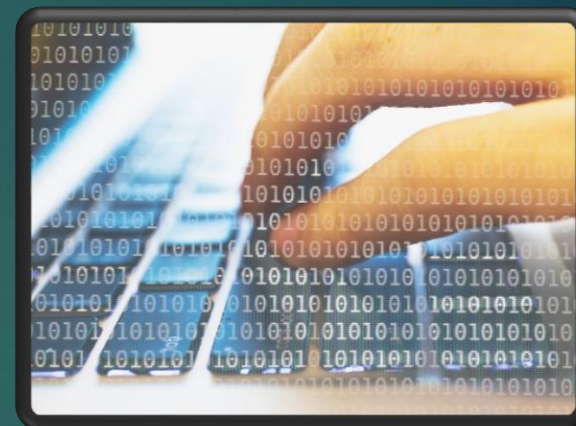
This book contains a good amount of Python tricks and tips which helps you to write better code. You already know how to solve congenital problems in three steps.



Reading



Figuring out



Practicing



Let's start right away
if you want to speed
up a bit to do a lot
of work.

Don't wait, get to
work!

Happy Pythoning.



System Requirements

▶ # Before you start check your python version. All the code run well above 3.6+ version.

```
import sys
!python --version
print(sys.version)
print(sys.version_info)
```

↳ Python 3.6.9
3.6.9 (default, Nov 7 2019, 10:44:02)
[GCC 8.3.0]
sys.version_info(major=3, minor=6, micro=9, releaselevel='final', serial=0)



Templating strings



```
print("I love %s with %s" % ("programming", "Python")) # old style

print("{} , {} and {}".format('python','javascript','c++')) #format with default order
print("{2}, {0} and {1}".format('python','javascript','c++')) # positional argument
print("{x}, {y} and {z}".format(x='python', y='javascript', z='c++')) # keyword argument
print("|{:<10}|{: ^10}|{:>10}|".format('Apple','Melon','Apricot')) #alignment

a=3.0; b=5.0; name="Python"
print(f'I have been developing {name} programs for {sum([a,b])} years.') # New Style
print(f'float format: {a*b:.2f} | {a*b:.5f} alignment:{a*b:>10}') # adjustment
```



```
I love programming with Python
python, javascript and c++
c++, python and javascript
python, javascript and c++
|Apple      | Melon    |  Apricot|
I have been developing Python programs for 8.0 years.
float format: 15.00 | 15.00000 alignment:      15.0
```



Concatenating

```
[ ] words = ['This', 'is', 'python', 'trick']  
    sentence = " ".join(words)  
    print(sentence)
```

```
☞ This is python trick
```




Splitting



```
sentence = "This is python trick"  
words = sentence.split()  
print(words)
```

```
☞ ['This', 'is', 'python', 'trick']
```



Reversing

```
[ ] word = "python"  
    words = ['This', 'is', 'python', 'trick']  
    print(word[::-1])  
    print(words[::-1])
```

```
☞ nohtyp  
   ['trick', 'python', 'is', 'This']
```



Printing out multiples of strings



```
print("Hello there! " * 2 + "Cool Python.. " * 2)  
for i in range(4) : print(f' Counter----> {i}' + " Python" * i )
```



```
Hello there! Hello there! Cool Python.. Cool Python..  
Counter----> 0  
Counter----> 1 Python  
Counter----> 2 Python Python  
Counter----> 3 Python Python Python
```



Removing unnecessary characters on strings

```
▶ name_with_space = "    Resul    "  
  name_with_slash = "////Resul////"  
  print(name_with_space.strip())  
  print( name_with_slash.strip('/') )
```



```
Resul  
Resul
```



Anagram Strings



```
from collections import Counter

str_1 = "listen"
str_2 = "silent"
cnt_1, cnt_2 = Counter(str_1), Counter(str_2)

if cnt_1 == cnt_2:
    print(f'Yeap!, {str_1} and {str_2} are anagram words.')
else:
    print('Not anagram')
```

☞ Yeap!, listen and silent are anagram words.



Palindrome Strings



```
my_string = "racecar"

if my_string == my_string[::-1]:
    print("yes, it is palindrome")
else:
    print("it is not palindrome")
```

```
☞ yes, it is palindrome
```



Frequency of elements in a List - Counter()



```
from collections import Counter
```

```
my_list = ['apple', 'apple', 'banana', 'banana', 'apple', 'apricot', 'fig', 'fig']  
count = Counter(my_list)
```

```
print(count)  
print(count['fig'])  
print(count.most_common(1))
```

```
☞ Counter({'apple': 3, 'banana': 2, 'fig': 2, 'apricot': 1})  
2  
[('apple', 3)]
```



Using itertools (combinations)



```
import itertools
friends = ['Team-A', 'Team-B', 'Team-C', 'Team-D']
list(itertools.combinations(friends, r=2))

#similarly you can use itertools.permutations()
```



```
[('Team-A', 'Team-B'),
 ('Team-A', 'Team-C'),
 ('Team-A', 'Team-D'),
 ('Team-B', 'Team-C'),
 ('Team-B', 'Team-D'),
 ('Team-C', 'Team-D')]
```



Unique elements in a string



```
my_string = "the quick brown fox jumps over the lazy dog"
```

```
temp_set = set(my_string)
```

```
new_string = ''.join(sorted(temp_set))
```

```
print(new_string)
```



```
abcdefghijklmnopqrstuvwxyz
```



Swapping values



```
x, y = 1, 2  
print(x, y)
```

```
x, y = y, x # Swapping only values, x != y  
print(x, y)
```



```
1 2  
2 1
```




Returning multiple values from a function



```
def my_func():  
    Id = 1 ; a = "Hakan"; b = "40" ; c = "Teacher"  
    return Id, a, b, c  
  
Id, name, age, job = my_func()  
  
print(f'{Id}th {name} is a {job} and {age} years old.')
```

☞ 1th Hakan is a Teacher and 40 years old.



Ternary operator



```
y = 30  
x = 10 if (y == 30) else 20  
print(x)
```

```
↳ 10
```

```
[29] for y in [40, -10]:  
      x = 10 if (y == 10) else 20 if (y == -10) else 40  
      print(x)
```

```
↳ 40  
   20
```



One line conditional if-else

```
▶ age =int(input("Age :"))  
print("old adults" if age >= 60 else "middle-aged" if 40 <= age < 60 else "young adult")
```

```
☞ Age :40  
middle-aged
```



Chained comparison



```
a = 4  
b = 8  
print(a < b < 10)  
print(a == b/2 < 10 > b)
```



```
True  
True
```



Chained function call



```
def multiply(a,b):  
    return a*b  
  
def add(a,b):  
    return a+b  
  
x = False  
print( (multiply if x else add)(3,5) )
```



8



Slicing a list

[start : stop : step]



```
num_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]  
steps = num_list[0::2]  
print(steps)
```

```
☐➔ [0, 2, 4, 6, 8]
```



unpacking list



```
num_list = [1, 2, 3, 4, 5, 6, 7, 8, 9]  
a, b, c, d = num_list[4:8]  
print(f'a = {a}, b = {b}, c = {c}, d = {d}')
```

```
a, *b, c, d = num_list  
print(f'a = {a}, b = {b}, c = {c}, d = {d}')
```

```
☞ a = 5, b = 6, c = 7, d = 8  
a = 1, b = [2, 3, 4, 5, 6, 7], c = 8, d = 9
```



unpacking list



```
my_list = ["Resul", "is", "a", "python", "developer"]  
print(' '.join(str(x) for x in my_list))  
  
print(*my_list, sep=' ') # unpackin the list with *
```



```
Resul is a python developer  
Resul is a python developer
```



Using *args & **kwargs arguments



```
db_tuples = (1,3,5,7,9)
```

```
def add_nums(*nums): # Note that *args is just a name.  
    sum = 0  
    for n in nums:  
        sum = sum + n  
    print("Sum:",sum , end=" | ")
```

```
add_nums(1,2,3)  
add_nums(4,5,6,7,8)  
add_nums(*db_tuples)
```

```
☞ Sum: 6 | Sum: 30 | Sum: 25 |
```



Using *args & **kwargs arguments



```
db_dict = {"name": "Kaan", "age": 14, "job": "student" }

def my_func(**kwargs): # Note that **kwargs is just a name
    for key, value in kwargs.items():
        print ("%s--> %s" %(key, value), end=" ", " ")
    print()

my_func(name = 'Hakan', age= 18, job='doctor')
my_func(**db_dict)
```



```
name--> Hakan, age--> 18, job--> doctor,
name--> Kaan, age--> 14, job--> student,
```




Copying List



```
# Fast way of copying -- shallow copy
list_1 = [1,2,3,4]
new_list = list_1 # new_list = list_1.copy() also smilar.

print(new_list)
print(new_list == list_1)
print(new_list is list_1)
print(id(new_list), "=", id(list_1))
```



```
[1, 2, 3, 4]
True
True
140138799980360 = 140138799980360
```



Copying List



```
# using list() or slicing.  
list_1 = [1,2,3,4]  
new_list = list_1[:] # new_list = list(list_1) also similar.  
  
print(new_list)  
print(new_list == list_1)  
print(new_list is list_1)  
print(id(new_list), "!=" , id(list_1))
```



```
[1, 2, 3, 4]  
True  
False  
140138789686856 != 140138791173000
```



Copying List



```
# using deepcopy() to copy nested list.  
from copy import deepcopy
```

```
list_1 = [[1,2],[3,4],[5,[6,7],8]]  
new_list = deepcopy(list_1)
```

```
print(new_list)  
print(new_list == list_1)  
print(new_list is list_1)  
print(id(new_list), "!=" , id(list_1))
```



```
[[1, 2], [3, 4], [5, [6, 7], 8]]  
True  
False  
140138789346440 != 140138799978504
```



List of list flattening



```
my_list = [[1, 2, 3], [4, 5], [6], [7, 8, 9]]  
sum(my_list, [])
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```



Min and max index in list



```
my_list = [4,5,1,3,2]
```

```
print(f'index of MIN. number in my_list is : {min(range(len(my_list)), key=my_list.__getitem__)}')  
print(f'index of MAX. number in my_list is : {max(range(len(my_list)), key=my_list.__getitem__)}')
```

```
index of MIN. number in my_list is : 2  
index of MAX. number in my_list is : 1
```




Using List Comprehensions

[expression for item in list if conditional]



```
# Old way of doing list
def square(a):
    return a**2

my_list = [1, 2, 3, 4, 5, 6, 7, 8]
new_list = []
for x in my_list:
    if x % 2 != 0:
        new_list.append(square(x))
print(new_list)
```

➞ [1, 9, 25, 49]



Using List Comprehensions

[expression for item in list if conditional]



Doing the same list with list comprehension is only one line.

```
mylist = [i**2 for i in range(8) if i % 2 != 0 ]  
print(mylist)
```

```
☞ [1, 9, 25, 49]
```



Using List Comprehensions

[expression for item in list if conditional]



```
another_comp_list = [i for i in range(50) if i%2==0 and i%5==0]  
print(another_comp_list)
```

```
☞ [0, 10, 20, 30, 40]
```



Using Dict&Set Comprehensions

{ expression for item in list if conditional }



```
set_comph = {i**2 for i in range(5)}  
print(set_comph)
```

```
dict_comph = {k: v**2 for k,v in zip('abcde',range(5))}  
print(dict_comph)
```

```
☐➤ {0, 1, 4, 9, 16}  
{'a': 0, 'b': 1, 'c': 4, 'd': 9, 'e': 16}
```



Merging Dictionaries



```
x = {'a': 1, 'b': 2}
y = {'c': 3, 'd': 4}
z = {'e': 5, 'f': 6}
merged = {**x, **y, **z}
print(merged)
```

```
☞ {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5, 'f': 6}
```




Iterating over a dictionary



```
dict1 = {'one': 1, 'two': 2, 'three': 3}
for key, value in dict1.items():
    print('{0:>5}: {1:<2}'.format(key, value))
```



```
one: 1
two: 2
three: 3
```



Inverting dictionary



```
dict1 = {'one': 1, 'two': 2, 'tree': 3, 'four': 4}
dict2 = { v: k for k, v in dict1.items() }
print(dict2)
```

```
☞ {1: 'one', 2: 'two', 3: 'tree', 4: 'four'}
```



Sorting dictionary by value



```
d = {"jacket":50, "bag":20, "hat":10, "tshirt":30, "shoes":40}  
print(sorted(d.items(), key=lambda x:x[1]))
```

```
[('hat', 10), ('bag', 20), ('tshirt', 30), ('shoes', 40), ('jacket', 50)]
```



Most frequent element in a list or string.

```
▶ numbers = [1, 2, 3, 4, 2, 2, 3, 1, 4, 4, 2, 4, 4]
  chars = "xxxyyzzzz"
  words = ["book", "phone", "e-reader", "tablet", "phone"]
  print(max(set(numbers), key = numbers.count))
  print(max(set(chars), key = chars.count))
  print(max(set(words), key = words.count))
```



```
4
z
phone
```



Dictionary get method



```
dict1 = dict(zip('abc', '123'))  
print(dict1)  
print(dict1.get("c", 4))  
print(dict1.get("d", 4)) # if key is not in dict return default value.  
# print(dict1["d"])    it results -> KeyError: 'd'
```



```
{'a': '1', 'b': '2', 'c': '3'}
```

```
3
```

```
4
```



Memory usage of an object.



```
import sys
obj = [i**2 for i in range(100)]
print(sys.getsizeof(obj))
```



912



Path of imported modules



```
import os, socket  
print(os)  
print(socket)
```



```
<module 'os' from '/usr/lib/python3.6/os.py'>  
<module 'socket' from '/usr/lib/python3.6/socket.py'>
```



getpass module



```
import getpass

users_name_pswd = {"root": "adminpassword"}

user = getpass.getuser()
password = getpass.getpass() # hides input passwords

if users_name_pswd[user] == password:
    print("You are logged in.")
else:
    print("Access Denied")
```



```
.....
You are logged in.
```



Using enumerate to get index/value pairs



```
my_list = ['item1', 'item2', 'item3', 'item4', 'item5']  
  
for index, value in enumerate(my_list):  
    print('{0}: {1}'.format(index+1, value))
```



```
1: item1  
2: item2  
3: item3  
4: item4  
5: item5
```



Zip(*iterables) Function



```
first_names = ["Hakan", "Kaan", "Asiye"]  
>>> last_names = ["Caliskan", "Aslan", "Yildiz"]  
>>> print([' '.join(x) for x in zip(first_names, last_names)])
```

```
['Hakan Caliskan', 'Kaan Aslan', 'Asiye Yıldız']
```



Zip(*iterables) Function

```
▶ id_list = [1, 2, 3, 4]
  name_list = ['Hakan', 'Kaan', 'Asiye', 'Resul']
  job_list = ['Doctor', 'Lawyer', 'Teacher', 'Engineer']

  print(list(zip(id_list, name_list, job_list)))

  for id, name, job in zip(id_list, name_list, job_list):
      print(f'{id}. {name} is a {job}.')
```

```
☞ [(1, 'Hakan', 'Doctor'), (2, 'Kaan', 'Lawyer'), (3, 'Asiye', 'Teacher'), (4, 'Resul', 'Engineer')]
1. Hakan is a Doctor.
2. Kaan is a Lawyer.
3. Asiye is a Teacher.
4. Resul is a Engineer.
```



Recursive function



```
def recursive_func(n):  
    # Base case: 1! = 1  
    if n == 1:  
        return 1  
  
    # Recursive case: n! = n * (n-1)!  
    else:  
        return n * recursive_func(n-1)  
  
print(recursive_func(9))
```



362880



Calculate the execution time



```
import time
# using time() method
start_time = time.time()
some_of_list = sum([num for num in range(1_000_000)])
end_time = time.time()

time_diff = (end_time - start_time)

print("Total process time: {0:.4} ms".format(time_diff))
```

```
☞ Total process time: 0.1248 ms
```



Calculate the execution time



```
import time
# using clock() method
start = time.clock()
sum((i * i for i in range(1, 1_000_000)))
stop = time.clock()
time_diff = stop - start
print(f"It took {time_diff:.4} Secs to execute this method")
```



```
It took 0.08644 Secs to execute this method
```




Files, directories of the current working directory.



```
import os
cwd = os.getcwd()
for dir_path, dir_names, file_names in os.walk(cwd):
    for f in file_names[:2]: # only two of files in every directory
        print(f)
```



```
.last_update_check.json
.last_survey_prompt.yaml
18.36.45.012839.log
18.36.32.968039.log
config_default
README.md
anscombe.json
```



Apply a function to every item in an iterable

`map(function, iterable)`



```
num_list = [1, 2, 3, 4, 5]
```

```
def square(num):  
    return num*num
```

```
print(list(map(square, num_list)))
```

```
☞ [1, 4, 9, 16, 25]
```



Lambda expressions-One line function.



```
square = lambda num: num * num  
print(square(5))
```

```
x = [1, 2, 3, 4, 5]  
print(list(map(lambda num: num * num, x)))
```



```
25  
[1, 4, 9, 16, 25]
```



Sorting with lambda function



```
my_list = [10, -20, 30, 40, -5]
```

```
print(sorted(my_list, key = lambda x : abs(x)))
```

```
[-5, 10, -20, 30, 40]
```



Reduce function

reduce(function, list)



```
from functools import reduce
```

```
product = reduce((lambda x, y: x * y), [1, 2, 3, 4])  
# smilar sum([1,4,9,16])
```

```
print(product)
```



24



Filter function

`filter(function, list)`



```
list_nums = list(range(100))
```

```
all_full_division_by_5 = list(filter(lambda num:num % 5 == 0, list_nums))
```

```
print(all_full_division_by_5)
```

```
[0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95]
```



Open two files



```
with open("first.txt") as my_file, open("second.txt") as second_file:  
    for line_number, (line1, line2) in enumerate(zip(my_file, second_file)):  
        print(line_number, (line1, line2))
```

```
0 (' Sentence in first file.\n', ' Sentence in second file.\n')  
1 (' Sentence in first file.\n', ' Sentence in second file.\n')
```



To query a request service



```
import requests
url = 'http://www.example.com/'
response = requests.get(url).text
print(response[:55])
```



```
<!doctype html>
<html>
<head>
    <title>Example Domain
```




Higher order functions

take a function as an argument, or return a function



```
def sum_of_nums(nums):  
    return sum(nums)  
  
def power_of_nums(nums):  
    return [i**2 for i in nums]  
  
def high_order_func(func, num_list):  
    return func(num_list)  
  
print(high_order_func(sum_of_nums, [1, 2, 3, 4]))  
print(high_order_func(power_of_nums, [1, 2, 3, 4]))
```



```
10  
[1, 4, 9, 16]
```



Generator (on the fly iteration)



```
def num_generator(num):  
    """ an infinite sequence generator that generates integers >= n """  
    while True:  
        yield num  
        num += 1  
  
plus_one = num_generator(4)      # starts at 4  
for _ in range(4):  
    print(next(plus_one))
```



```
4  
5  
6  
7
```



Using coroutine and decorators



```
#@title Power function. { vertical-output: true }
import math
""" This program has two function first one is couroutine
and the latter is decorator.
It produces the power of given number """
def coroutine(gen_fn):
    def inner(*args, **kwargs):
        gen = gen_fn(*args, **kwargs)
        next(gen)
        return gen
    return inner

@coroutine
def power(p):
    result = None
    while True:
        received = yield result
        result = math.pow(received, p)

squares = power(2); cubes = power(3); quadro = power(4); quintic = power(5)
for power in [squares, cubes, quadro, quintic]: print(power.send(4))
```

Power function.

```
➤ 16.0
   64.0
   256.0
   1024.0
```



Error handling "try-except-else-finally" blocks



```
try:
    file = open("file.txt")
    new_file = file.readline()
except FileNotFoundError:          # exception raised when file.txt is absent.
    print('File not found, and a new "file.txt" file is created.')
    file = open("file.txt", 'w')
else:                             # this runs only in the absence of exceptions.
    new_file = open("file.txt")
    data = new_file.read()
finally:                          # this finally part always run.
    file.close()
```

☞ File not found, and a new "file.txt" file is created.



References

- ▶ [3.8.1 Documentation](#)
- ▶ [The Python Language Reference](#)
- ▶ <https://stackoverflow.com/>
- ▶ experimenting patiently..



*Congratulations
happy pythoning!*

THANK YOU

RESUL ÇALIŞKAN
DATA SCIENTIST

R.CALISKAN@OUTLOOK.COM

[HTTPS://GITHUB.COM/THEWORLDOFCODING/PYTHON-TIPS-AND-TRICKS](https://github.com/theworldofcoding/python-tips-and-tricks)