PYTHON
TIPS & TRICKS

A COLLECTION OF 100 BASIC AND INTERMEDIATE TIPS AND TRICKS

Benjamin Bennett Alexander

PYTHON

TIPS & TRICKS

A COLLECTION OF

100 BASIC & INTERMEDIATE TIPS AND TRICKS

Benjamin Bennett Alexander

Feedback and Reviews

I welcome and appreciate your feedback and reviews. Feedback help independent writers reach more people. So please consider leaving feedback on the platform you got this book from. Send queries to: benjaminbennettalexander@gmail.com.

Table of Contents

About this Book

This book is a collection of Python tips and tricks. I have put together 100 Python tips and tricks that you may find helpful if you are learning Python. The tips cover mainly basic and intermediate levels of Python.

In this book you will find tips and tricks on:
- How to print horizontally using the print function
- How to use list comprehensions to make code my concise
- How to update a dictionary using dictionary comprehensions
- How to merge dictionaries
- Swapping variables
- Merging PDF files
- Creating DataFrames using pandas
- Correct spelling errors with Python
- Censor profanity using Python
- How to reset recursion limit
- Extracting zip files using Python
- Converting text to handwriting
- Taking screen shots with Python
- Generate dummy data with Python
- Finding permutations of a string
- Using sort method and sorted function to sort iterables
- Writing CSV files and many more

To fully benefit, please try your best to write the code down and run it. It is important that you try to understand how the code works. Modify and improve the code. Do not be afraid to experiment.

By the end of it all, I hope that the tips and tricks will help you level up your Python skills and knowledge.

Let's get started.

1 Printing Horizontally

When looping over an iterable, the print function prints each item on a new line. This is because the print function has a parameter called end. By default, the end parameter has an escape character (end = "\n"). Now, to print horizontally, we need to remove the escape character and replace it with an empty string (end = ""). In the code below, take note of the space between the commas (" "); this is to ensure that the numbers are printed with spaces between them. If we remove the space (""), the numbers will be printed like this: 1367. Here is the code to demonstrate that:

```
list1 = [1, 3, 6, 7]

for number in list1:
    print(number, end= " ")
```

```
Output:
1 3 6 7
```

The print function has another parameter called **sep**. We use *sep* to specify how the output is separated. Here is an example:

```
print('12','12','1990', sep='/')
```

```
Output:
12/12/1990
```

2 | Merging Dictionaries

If you have two dictionaries that you want to combine, you can do so using two easy methods. You can use the merge (|) operator or the (**) operator. Below, we have two dictionaries, a and b. We are going to use these two methods to combine the dictionaries. Here are the codes below:

Method 1

Using the merge (|) operator.

```
name1 = {"kelly": 23,
         "Derick": 14, "John": 7}
name2 = {"Ravi": 45, "Mpho": 67}

names = name1 | name2
print(names)
Output:
{'kelly': 23, 'Derick': 14, 'John': 7, 'Ravi': 45,
'Mpho': 67}
```

Method 2

Using the merge (**) operator. With this operator, you need to put the dictionaries inside the curly brackets.

```
name1 = {"kelly": 23,
         "Derick": 14, "John": 7}
name2 = {"Ravi": 45, "Mpho": 67}

names = {**name1, **name2}
print(names)
Output:
{'kelly': 23, 'Derick': 14, 'John': 7, 'Ravi': 45,
'Mpho': 67}
```

3 | Calendar with Python

Did you know that you can get a calendar using Python?

Python has a built-in module called **calendar**. We can import this module to print out the calendar. We can do a lot of things with calendar.

Let's say we want to see April 2022; we will use the *month* class of the calendar module and pass the year and month as arguments. See below:

```
import calendar

month = calendar.month(2022, 4)
print(month)
```

```
Output:
```

```
 April 2022
Mo Tu We Th Fr Sa Su
             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 26 27 28 29 30
```

There are so many other things you can do with the calendar. For example, you can use it to check if a given year is a leap year or not. Let's check if 2022 is a leap year.

```
import calendar

month = calendar.isleap(2022)
print(month)
```

```
Output:
False
```

4 | Get Current Time and Date

The below code demonstrates how you can get the current time using the *datetime()* module. The *strftime()* method formats time for the desired output. This code breaks down how you can use the datetime module with the *strftime()* method to get a formatted string of time in hours, minutes, and seconds format.

```
from datetime import datetime

time_now = datetime.now().strftime('%H:%M:%S')
print(f'The time now is {time_now}')
Output:
The time now is 17:53:19
```

What if we want to return today's date? We can use the date class from the datetime module. We use the *today()* method. See below:

```
from datetime import date

today_date = date.today()
print(today_date)
Output:
2022-07-21
```

5 | Sort a List in Descending Order

The *sort()* method will sort a list in ascending order (by default). For the sort method to work, the list should have the same type of objects. You cannot sort a list mixed with different data types, such as integers and strings. The *sort()* method has a parameter called reverse; to sort a list in descending order, set reverse to True.

```
list1 = [2, 5, 6, 8, 1, 8, 9, 11]

list1.sort(reverse=True)
print(list1)
Output:
[11, 9, 8, 8, 6, 5, 2, 1]
```

Remember, *sort()* is strictly a list method. You cannot use it to sort a **set**, a **tuple,** a **string,** or a **dictionary**.

The *sort()* method does not return a new list; it sorts an existing list. If you try to create a new object using the *sort()* method, it will return None. See below:

```
list1 = [2, 5, 6, 8, 1, 8, 9, 11]
list2 = list1.sort(reverse=True)
print(list2)
Output:
None
```

6 | Swapping Variables

In Python, you can swap variables once they have been assigned to objects. Below, we initially assign 20 to x and 30 to y, but then we swap them: x becomes 30 and y becomes 20.

```
x, y = 20, 30
x, y = y, x

print('x is', x)
print('y is', y)
```

```
Output:
x is 30
y is 20
```

In Python, we can also use the XOR (exclusive or) operator to swap variables. This is a three-step method. In the example below, we are swapping the values of x and y.

```
x = 20
y = 30

# step one
x ^= y
# step two
y ^= x
# step three
x ^= y

print(f'x is: {x}')
print(f'y is: {y}')
```

```
Output:
x is: 30
y is: 20
```

7 | Counting Item Occurrences

If you want to know how many times an item appears in an iterable, you can use the *counter()* class from the collection module. The *counter()* class will return a dictionary of how many times each item appears in an iterable. Let's say we want to know how many times the name Peter appears in the following list; here is how we can use the *counter()* class of the collections module.

```
from collections import Counter

list1 = ['John','Kelly', 'Peter', 'Moses', 'Peter']

count_peter = Counter(list1).get("Peter")

print(f'The name Peter appears in the list '
      f'{count_peter} times.')
```

```
Output:
The name Peter appears in the list 2 times.
```

Another way you can do it is by using a normal for loop. This is the naive way. See below:

```
list1 = ['John','Kelly', 'Peter', 'Moses', 'Peter']

# Create a count variable
count = 0
for name in list1:
    if name == 'Peter':
        count +=1
print(f'The name Peter appears in the list'
      f' {count} times.')
```

```
Output:
The name Peter appears in the list 2 times.
```

8 | Flatten a Nested List

I will share with you three (3) ways you can flatten a list. The first method employs a *for* loop, the second employs the *itertools* module, and the third employs *list comprehension*.

```
list1 = [[1, 2, 3],[4, 5, 6]]

newlist = []
for list2 in list1:
    for j in list2:
        newlist.append(j)
print(newlist)
Output:
[1, 2, 3, 4, 5, 6]
```

Using itertools

```
import itertools

list1 = [[1, 2, 3],[4, 5, 6]]

flat_list = list(itertools.chain.from_iterable(list1))
print(flat_list)
Output:
[1, 2, 3, 4, 5, 6]
```

Using list comprehension

If you do not want to import *itertools* or the normal for loop, you can just use list comprehension.

```
list1 = [[1, 2, 3], [4, 5, 6]]

flat_list= [i for j in list1 for i in j]
print(flat_list)
Output:
[1, 2, 3, 4, 5, 6]
```

9 | Index of the Biggest Number

Using the *enumerate()*, *max()*, and *min()* functions, we can find the index of the biggest and the smallest number in a list. The max function is a high-order function; it takes another function as an argument.

In the code below, the *max()* function takes the list and the lambda function as arguments. We add the *enumerate*() function to the list so it can return both the number in the list and its index (a tuple). We set the count in the enumerate function to start at position zero (0). The lambda function tells the function to return a tuple pair of the maximum number and its index.

Find the index of the largest number

```
x = [12, 45, 67, 89, 34, 67, 13]

max_num = max(enumerate(x, start=0),
                  key = lambda x: x[1])
print('The index of the largest number is',
      max_num[0])
Output:
The index of the biggest number is 3
```

Finding the index of the smallest number

This is similar to the *max()* function, only that we are now using the *min()* function.

```
x = [12, 45, 67, 89, 34, 67, 13]

min_num = min(enumerate(x, start=0),
                  key = lambda x : x[1])
print('The index of the smallest number is',
      min_num[0])
Output:
The index of the smallest number is 0
```

10 | Absolute Value of a Number

Let's say you have a negative number and you want to return the absolute value of that number; you can use the *abs()* function. The Python *abs()* function is used to return the absolute value of any number (positive, negative, and complex numbers). Below, we demonstrate how we can return a list of absolute numbers from a list of numbers that have negative and positive numbers. We use list comprehension.

```
list1 = [-12, -45, -67, -89, -34, 67, -13]

print([abs(num) for num in list1])
Output:
[12, 45, 67, 89, 34, 67, 13]
```

You can also use the *abs()* function on a floating number, and it will return the absolute value. See below:

```
a = -23.12
print(abs(a))
Output:
23.12
```

When you use it on a complex number, it returns the magnitude of that number. See below:

```
complex_num = 6 + 3j
print(abs(complex_num))
Output:
6.708203932499369
```

11 | Adding a Thousand Separator

If you are working with large figures and you want to add a separator to make them more readable, you can use the *format()* function. See the example below:

```
a = [10989767, 9876780, 9908763]

new_list =['{:,}'.format(num) for num in a]
print(new_list)
Output:
['10,989,767', '9,876,780', '9,908,763']
```

We can also use f-strings to add a thousand separators. Notice below that instead of using a comma (,) as a separator, we are using an underscore (_).

```
a = [10989767, 9876780, 9908763]

new_list =[f"{num:_}" for num in a]
print(new_list)
Output:
['10_989_767', '9_876_780', '9_908_763']
```

Have you noticed that we are using list comprehension to add the separator on both occasions? Cool thing, right? 😊.

Later on, we'll look at another method for adding a thousand separator.

12 | Startswith and Endswith Methods

The *startswith()* and *endswith()* are string methods that return True if a specified string starts with or ends with a specified value.

Let's say you want to return all the names in a list that start with "a."; here is how you would use *startswith()* to accomplish that.

Using startswith()

```
list1 = ['lemon','Orange',
         'apple', 'apricot']

new_list = [i for i in list1 if i.startswith('a')]
print(new_list)
Output:
['apple', 'apricot']
```

Using endswith()

```
list1 = ['lemon','Orange',
         'apple', 'apricot']

new_list = [i for i in list1 if i.endswith('e')]
print(new_list)
Output:
['Orange', 'apple']
```

Notice that for both examples above, we are using list comprehension.

13 | Nlargest and Nsmallest

Let's say you have a list of numbers and you want to return the five largest numbers or the five smallest numbers from that list. Normally, you can use the *sorted()* function and list slicing, but they only return a single number.

```
def sort_list(arr: list):
    a = sorted(arr, reverse=True)
    return a[:5]

results = [12, 34, 67, 98, 90, 68, 55, 54, 64, 35]

print(sort_list(results))
Output:
[98, 90, 68, 67, 64]
```

This is cool, but slicing does not make code readable. There is a Python built-in module that you can use that will make your life easier. It is called *heapq*. With this module, we can easily return the 5 largest numbers using the *nlargest* method. The method takes two arguments: the iterable object and the number of numbers we want to return. Below, we pass 5, because we want to return the five largest numbers from the list.

Using nlargest

```
import heapq

results = [12, 34, 67, 98, 90, 68, 55, 54, 64, 35]
print(heapq.nlargest(5, results))
Output:
[98, 90, 68, 67, 64]
```

Using nsmallest

```
Import headq

results = [12, 34, 67, 98, 90, 68, 55, 54, 64, 35]
print(headq.nsmallest(5, results))
Output:
[12, 34, 35, 54, 55]
```

14 | Checking for Anagram

You have two strings; how would you go about checking if they are anagrams?

If you want to check if two strings are anagrams, you can use *counter*() from the *collections* module. The *counter()* supports equality tests. We can basically use it to check if the given objects are equal. In the code below, we are checking if *a* and *b* are anagrams.

```
from collections import Counter

a = 'lost'
b = 'stol'
print(Counter(a)==Counter(b))
Output:
True
```

We can also use the *sorted()* function to check if two strings are anagrams. By default, the *sorted()* function will sort a given string in ascending order. So, when we pass strings as arguments to the sorted function for equality tests, first the strings are sorted, then compared. See the code below:

```
a = 'lost'
b = 'stol'

if sorted(a)== sorted(b):
    print('Anagrams')
else:
    print("Not anagrams")
Output:
Anagrams
```

15 | Checking Internet Speed

Did you know that you can check internet speed with Python?

There is a module called *speedtest* that you can use to check the speed of your internet. You will have to install it with pip.

pip install speedtest-cli

Since the output of speedtest is in bits, we divide it by 8000000 to get the results in mb. Go on, test your internet speed using Python.

Checking download speed

```
import speedtest

d_speed = speedtest.Speedtest()
print(f'{d_speed.download()/8000000:.2f}mb')
```

```
Output:
213.78mb
```

Checking upload speed

```
import speedtest

up_speed = speedtest.Speedtest()
print(f'{up_speed.upload()/8000000:.2f}mb')
```

```
Output:
85.31mb
```

16 | Python Reserved keywords

If you want to know the reserved keywords in Python, you can use the *help()* function. **Remember, you cannot use any of these words as variable names.** Your code will generate an error.

```
print(help('keywords'))
```

Output:

```
Here is a list of the Python keywords.  Enter any
    keyword to get more help.

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

None
```

17 | Properties and Methods

If you want to know the properties and methods of an object or module, use the *dir()* function. Below, we are checking for the properties and methods of a string object. You can also use *dir()* to check the properties and methods of modules. For instance, if you wanted to know the properties and methods of the *collections* module, you could import it and pass it as an argument to the function print(dir(collections)).

```
a = 'I love Python'
print(dir(a))
Output:
['__add__', '__class__', '__contains__', '__delattr__',
'__dir__', '__doc__', '__eq__', '__format__', '__ge__',
'__getattribute__', '__getitem__', '__getnewargs__',
'__gt__', '__hash__', '__init__', '__init_subclass__',
'__iter__', '__le__', '__len__', '__lt__', '__mod__',
'__mul__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__rmod__', '__rmul__',
'__setattr__', '__sizeof__', '__str__',
'__subclasshook__', 'capitalize', 'casefold', 'center',
'count', 'encode', 'endswith', 'expandtabs', 'find',
'format', 'format_map', 'index', 'isalnum', 'isalpha',
'isascii', 'isdecimal', 'isdigit', 'isidentifier',
'islower', 'isnumeric', 'isprintable', 'isspace',
'istitle', 'isupper', 'join', 'ljust', 'lower',
'lstrip', 'maketrans', 'partition', 'removeprefix',
'removesuffix', 'replace', 'rfind', 'rindex', 'rjust',
'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines',
'startswith', 'strip', 'swapcase', 'title', 'translate',
'upper', 'zfill']
```

18 | Open a Website Using Python

Did you know that you can open a website using a Python script?

To open a website using Python, import the *webbrowser* module. This is a built-in module, so you don't have to install anything. Below, we are trying to open google.com using the *open()* method of the module. You can do this with any website if you know its URL.

```
import webbrowser

url = "https://www.google.com/"
open_web = webbrowser.open(url)
print(open_web)
```

You can also specify if you want to open a new tab in a browser or a new browser window. See the code below:

```
import webbrowser

url = "https://www.google.com/"

# This opens a new tab in your browser
webbrowser.open_new_tab(url)

# This opens a new browser window
webbrowser.open_new(website)
```

19 | Most Frequent in a String

Let's say you have a string and you want to find the most frequent element in the string; you can use the *max()* function. The *max()* function will count the items in the string and return the item that appears the most. You just have to pass the string *count* method to the key parameter. Let's use the string below to demonstrate this.

```
a = 'fecklessness'

most_frequent = max(a, key = a.count)
print(f'The most frequent letter is, '
      f'{most_frequent}')
Output
The most frequent letter is, s
```

Now, if there is more than one most frequent item in the string, the max() function will return the element that comes first alphabetically. For example, if the string above had 4 Fs and 4 Ss, the max function would return "f" instead of "s" as the most frequent element because "f" comes first alphabetically.

We can also use the **Counter** class from the collections module. The *most_common*() method of this class will count how many times each element appears in the list, and it will return all the elements and their counts, as a list of tuples. Below, we pass the parameter (1) because we want it to return the number one most common element of the list. If we pass (2), it will return the two most common elements.

```
import collections

a = 'fecklessness'

print(collections.Counter(a).most_common(1))
Output:
('s', 4)
```

20 | Memory Size Check

Do you want to know how much memory an object is consuming in Python?

The *sys* module has a method that you can use for such a task. Here is a code to demonstrate this. Given the same items, which one is more memory efficient among a set, a tuple, and a list? Let's use the sys module to find out.

```
import sys

a = ['Love', 'Cold', 'Hot', 'Python']
b = {'Love', 'Cold', 'Hot', 'Python'}
c =  ('Love', 'Cold', 'Hot', 'Python')

print(f'The memory size of a list is '
      f'{sys.getsizeof(a)} ')

print(f'The memory size of a set is '
      f'{sys.getsizeof(b)} ')

print(f'The memory size of a tuple is '
      f'{sys.getsizeof(c)} ')
```

```
Output:
The memory size of a list is 88
The memory size of a set is 216
The memory size of a tuple is 72
```

As you can see, lists and tuples are more space efficient than sets.

21 | Accessing Dictionary Keys

How do you access keys in a dictionary? Below are three different ways you can access the keys of a dictionary.

1. Using set comprehension

Set comprehension is similar to list comprehension. The difference is that it returns a set.

```
dict1 = {'name': 'Mary', 'age': 22,
         'student':True,'Country': 'UAE'}

print({key for key in dict1.keys()})
Output:
{'age', 'student', 'Country', 'name'}
```

2. Using the set() function

```
dict1 = {'name': 'Mary', 'age': 22,
         'student':True,'Country': 'UAE'}

print(set(dict1))
Output:
{'name', 'Country', 'student', 'age'}
```

3. Using the sorted() function

```
dict1 = {'name': 'Mary', 'age': 22,
         'student':True,'Country': 'UAE'}

print(sorted(dict1))
Output:
['Country', 'age', 'name', 'student']
```

22 | Iterable or Not

Question: How do you confirm if an object is iterable using code?

This is how you can use code to check if an item is *iterable* or not. We are using the *iter()* function. When you pass an item that is not iterable as an argument to the *iter()* function, it returns a TypeError. Below, we write a short script that checks if a given object is an iterable.

```
arr = ['i', 'love', 'working', 'with', 'Python']


try:
    iter_check = iter(arr)
except TypeError:
    print('Object a not iterable')
else:
    print('Object a is iterable')


# Check the second object
b = 45.7

try:
    iter_check = iter(b)
except TypeError:
    print('Object b is not iterable')
else:
    print('Object b is iterable')
```

```
Output:
Object a is iterable
Object b is not iterable
```

23 | Sorting a List of Tuples

You can sort a list of tuples using the *itemgetter()* class of the operator module. The *itemgetter()* function is passed as a key to the *sorted()* function. If you want to sort by the first name, you pass the index (0) to the *itemgetter()* function. Below are different ways you can use *itemgetter()* to sort the list of tuples.

```
from operator import itemgetter

names = [('Ben','Jones'),('Peter','Parker'),
         ('Kelly','Isa')]

#Sort names by first name
sorted_names = sorted(names,key=itemgetter(0))
print('Sorted by first name',sorted_names)

# sort names by last name
sorted_names = sorted(names,key=itemgetter(1))
print('Sorted by last name',sorted_names)

# sort names by first name, then last name
sorted_names = sorted(names,key=itemgetter(0,1))
print('Sorted by last name & first name',sorted_names)
```

Output:

```
Sorted by first name [('Ben', 'Jones'), ('Kelly',
   'Isa'), ('Peter', 'Parker')]

Sorted by last name [('Kelly', 'Isa'), ('Ben', 'Jones'),
   ('Peter', 'Parker')]

Sorted by last name & first name [('Ben', 'Jones'),
   ('Kelly', 'Isa'), ('Peter', 'Parker')]
```

24 | Sort List with Sorted & Lambda

The *sorted()* function is a high-order function because it takes another function as a parameter. Here, we create a *lambda* function that is then passed as an argument to the *sorted()* function key parameter. By using a negative index [-1], we are telling the *sorted()* function to sort the list in descending order.

```
list1 = ['Mary', 'Peter', 'Kelly']

a = lambda x: x[-1]
y = sorted(list1, key=a)
print(y)
Output:
['Peter', 'Mary', 'Kelly']
```

To sort the list in ascending order, we can just change the index to [:1]. See below:

```
list1 = ['Mary', 'Peter', 'Kelly']

a = lambda x: x[:1]
y = sorted(list1, key=a)
print(y)
Output:
['Kelly', 'Mary', 'Peter']
```

Another easy way to sort the list in ascending order would be to use just the *sorted()* function. By default, it sorts an iterable in ascending order. Since the key parameter is optional, we just leave it out.

```
list1 = ['Mary', 'Peter', 'Kelly']

list2 = sorted(list1)
print(list2)
Output
['Kelly', 'Mary', 'Peter']
```

25 | Access News Using Python

You can do a lot of things with Python, even read the news. You can access the news using the Python newspapers module. You can use this module to scrape newspaper articles. First, install the module.

pip install newspaper3k

Below we access the title of the article. All we need is the article URL.

```
from newspaper import Article

news = Article("https://indianexpress.com/article/"
               "technology/gadgets/"
               "apple-discontinues-its-last-ipod-model-7910720/")

news.download()
news.parse()
print(news.title)
```

```
Output
End of an Era: Apple discontinues its last iPod model
```

We can also access the article text, using the text method.

```
news = Article("https://indianexpress.com/article/"
               "technology/gadgets/"
               "apple-discontinues-its-last-ipod-model-7910720/")

news.download()
news.parse()
print(news.text)
```

```
Output:
Apple Inc.'s iPod, a groundbreaking device that upended the
music and electronics industries more than two decades ago...
```

We can also access the article publication date.

```
news = Article("https://indianexpress.com/article/"
               "technology/gadgets/"
               "apple-discontinues-its-last-ipod-model-7910720/")

news.download()
news.parse()
print(news.publish_date)
```

```
Output:
2022-05-11 09:29:17+05:30
```

26 | A List of Tuples with Enumerate

Since enumerate counts (adds a counter) the items it loops over, you can use it to create a list of tuples. Below, we create a list of tuples of days in a week from a list of days. Enumerate has a parameter called "start." Start is the index at which you want the count to begin. By default, the start is zero (0).

Below, we have set the start parameter to one (1). You can start with any number you want.

```
days = ["Sunday", "Monday", "Tuesday", "Wednesday",
        "Thursday", "Friday", "Saturday"]

days_tuples = list(enumerate(days, start=1))
print(days_tuples)
Output:
[(1,   'Sunday'),   (2,   'Monday'),   (3,   'Tuesday'),   (4,
'Wednesday'), (5, 'Thursday'), (6, 'Friday'), (7, 'Saturday')]
```

It is also possible to create the same output with a *for loop*. Let's create a function to demonstrate this.

```
def enumerate(days, start= 1):
    n = start
    for day in days:
        yield n, day
        n += 1


days = ["Sunday", "Monday", "Tuesday", "Wednesday",
        "Thursday", "Friday", "Saturday"]

print(list(enumerate(days)))
Output:
[(1, 'Sunday'), (2, 'Monday'), (3, 'Tuesday'), (4, 'Wednesday'), (5,
'Thursday'), (6, 'Friday'), (7, 'Saturday')]
```

27 | Assertion

We can use the assert statement to check or debug your code. The assert statement will catch errors early on in the code. The assert statement takes two arguments: a condition and an optional error message. Here is the syntax below:

assert <condition>, [error message]

The condition returns a Boolean value of either True or False. The error message is the message we want to be displayed if the condition is False.

Below, we insert an assert statement in the code. This code takes a list of names and returns all the names in lowercase letters. We expect all the items in the list to be strings, so we use the insert statement to debug for non-string entries. The insert statement will check if all the items in the list are of type *str*. If one of the items is not a string, it will evaluate to False. It will halt the program and throw an *AssertionError*. It will display the error message that "**non-string items are in the list**." If all the items are strings, it will evaluate to True and run the rest of the code. The code returns an error because the fourth name in the list of names is not a string.

```
name = ["Jon","kelly", "kess", "PETR", 4]

lower_names = []
for item in name:
    assert type(item) == str, 'non-string items in the list'
    if item.islower():
        lower_names.append(item)

print(lower_names)
Output:
AssertionError: non-string items in the list
```

If we remove the non-string item from the list, the rest of the code runs.

```
name = ["Jon","kelly", "kess", "PETR"]

lower_names = []
for item in name:
    assert type(item) == str, 'non-string items in the list'
    if item.islower():
        lower_names.append(item)

print(lower_names)
Output:
['kelly', 'kess']
```

28 | Print Colored Texts

Did you know that you can add color to your code using Python and ANSI escape codes? Below, I created a class of codes and applied it to the code that I printed out.

```
class Colors():
    Black = '\033[30m'
    Green = '\033[32m'
    Blue = '\033[34m'
    Magenta = '\033[35m'
    Red = '\033[31m'
    Cyan = '\033[36m'
    White = '\033[37m'
    Yellow = '\033[33m'


print(f'{Colors.Red} Warning: {Colors.Green} '
      f'Love Don\'t live here anymore')
```

Output:

```
Warning:  Love Don't live here anymore
```

29 | Find Index Using Enumerate

The simplest way to access the index of items in an iterable is by using the *enumerate()* function. By default, the enumerate function returns the element and its index. We can basically use it to loop over an iterable, and it will return a counter of all the elements in the iterable.

Let's say we want to find the index of the letter "n" in *str1* below. Here is how we can use the enumerate function to achieve that:

```
str1 = 'string'

for index, value in enumerate(str1):
    if value =='n':
        print(f"The index of n is {index}")
Output
The index of 'n' is 4
```

If we want to print all the elements in the string and their indexes, here is how we can use enumerate.

```
str1 = 'string'
for i, j in enumerate(str1):
    print(f"Index: {i}, Value: {j}")
Output:
Index: 0, Value: s
Index: 1, Value: t
Index: 2, Value: r
Index: 3, Value: i
Index: 4, Value: n
Index: 5, Value: g
```

If you don't want to use a for loop, you can use enumerate with the list function, and it will return a list of tuples. Each tuple will have a value and its index.

```
str1 = 'string'
str_counta = list(enumerate(str1))
print(str_counta)
Output:
[(0, 's'), (1, 't'), (2, 'r'), (3, 'i'), (4, 'n'), (5, 'g')]
```

30 | Create Class Using Type Function

The *type()* function is usually used to check the type of an object. However, it can also be used to create classes dynamically in Python.

Below I have created two classes, the first one using the class keyword and the second one using the *type()* function. You can see that both methods achieve the same results.

```
# Creating dynamic class using the class keyword
class Car:
    def __init__(self, name, color):
        self.name = name
        self.color = color

    def print_car(self):
        return f'The car is {self.name} ' \
               f'and its {self.color} in color'

car1 = Car('BMW', 'Green')
print(car1.print_car())

# Creating dynamic class using the type keyword
def cars_init(self, name ,color):
    self.name = name
    self.color = color

Cars = type("Car",(),
            {'__init__': cars_init,
            'print_car':lambda  self:
            f'The car is {self.name} '
            f'and its {self.color} in color'})

car1 = Cars("BMW", "Green")
print(car1.print_car())
Output:
The car is BMW and its Green in color
The car is BMW and its Green in color
```

31 | Checking if a String is Empty

The simple way to check if a given string is empty is to use the *if statement* and the ***not*** operator, which will return a Boolean value. Empty strings in Python evaluate to False, and strings that are not empty evaluate to True. The ***not*** operator returns True if something is False and False if something is True. Since an empty string evaluates to False, the ***not*** operator will return True. Below, if a string is empty, the *if* statement will evaluate to True, so that part of the code will run. If a string is not empty, the *else* statement will run.

```
# Empty string
str1 = ''

if not str1:
    print('This string is empty')
else:
    print('This string is NOT empty')
Output:
This string is empty
```

Example 2

Let's try to insert something into the string now.

```
# Not empty string
str2 = 'string'

if not str1:
    print('This string is empty')
else:
    print('This string is NOT empty')
Output:
This string is NOT empty
```

32 | Flatten Nested List

What is a nested list? A nested list is a list that has another list as an element (a list inside a list). You can flatten a nested list using the *sum()* function. Note that this will work on a two-dimensional nested list.

```
nested_list = [[2, 4, 6],[8, 10, 12]]

new_list = sum(nested_list,[])
print(new_list)
Output:
[2, 4, 6, 8, 10, 12]
```

Please note that this is not the most efficient way to flatten a list. It is not easy to read. But it's still pretty cool to know, right? 😊

Using reduce function

Here is another cool trick you can use to flatten a two-dimensional list. This time we use *reduce* from functools module. This is another high order function in Python.

```
from functools import reduce

nested_list = [[2, 4, 6], [8, 10, 12]]

new_list = reduce(lambda x, y: x+y, nested_list)
print(new_list)
Output:
[2, 4, 6, 8, 10, 12]
```

33 | Checking if a File Exists

Using the OS module, you can check if a file exists. The os module has an *exists()* function from the path() class that returns a Boolean value. If a file exists, it will return True; if not, it will return False. When working with files, it is important to check if a file exists before trying to run it to avoid generating errors. Let's say you want to delete or remove a file with Python; if the file does not exist, your code will generate an error. See the following example:

```
import os

os.remove("thisfile.txt")
Output:
FileNotFoundError: [WinError 2] The system cannot find the file
specified: 'thisfile.txt'
```

To avoid this error, we have to check if the file exists before removing it. We will pass the file path or file name as an argument to os.path.exists(). If the file is in the same folder as your Python file, we can pass the file name as an argument.

In the code below, we use the file name as an argument because we assume the file is in the same folder as the Python file. You can see from the output that even though the file does not exist, our code does not generate an error.

```
import os.path

file = os.path.exists('thisfile.txt')

if file:
    os.remove("thisfile.txt")
else:
    print('This file does Not exist')
Output:
This file does Not exist
```

34 | Set Comprehension

Set comprehension is similar to list comprehension; the only difference is that it returns a set and not a list. Instead of square brackets, set comprehension uses curly brackets. This is because sets are enclosed in curly brackets. You can use set comprehension on an iterable (list, tuple, set, etc.).

Let's say we have a list of uppercase strings and we want to convert them into lowercase strings and remove duplicates; we can use set comprehension. Since sets are not ordered, the order of the items in the iterable will be changed. Sets do not allow duplicates, so only one "PEACE" will be in the output set.

```
list1 = ['LOVE', 'HATE', 'WAR', 'PEACE', 'PEACE']

set1 = {word.lower()for word in list1}
print(set1)
Output:
{'love', 'peace', 'war', 'hate'}
```

Here is another way we can use set comprehension. Let's say we have a list of numbers and we want to return all the numbers from the list that are divisible by 2 and remove duplicates at the same time. Here is the code below: Remember, sets do not allow duplicates, so it will remove all the numbers that appear more than once.

```
arr = [10, 23, 30, 30, 40, 45, 50]

new_set = {num for num in arr if num % 2 == 0}
print(new_set)
Output:
{40, 10, 50, 30}
```

35 | Python *args and **Kwargs

When you are not sure how many arguments you will need for your function, you can pass *args (non-keyword arguments) as a parameter. The * notation tells Python that you are not sure how many arguments you need, and Python allows you to pass in as many arguments as you want. Below, we calculate the average with different numbers of arguments. First, we pass three (3) numbers as arguments. Then we pass six numbers as arguments. The *args make functions more flexible when it comes to arguments.

```
def avg(*args):
    avg1 = sum(args)/len(args)
    return f'The average is {avg1:.2f}'

print(avg(12, 34, 56))
print(avg(23,45,36,41,25,25))
```

Output:

```
The average is 34.00
The average is 32.50
```

When you see ****kwargs** (keyword arguments) as a parameter, it means the function can accept any number of arguments as a dictionary (arguments must be in key-value pairs). See the example below:

```
def myFunc(**kwargs):
    for key, value in kwargs.items():
        print(f'{key} = {value}')
    print('\n')

myFunc(Name = 'Ben',Age = 80, Occupation ='Engineer')
```

Output:

```
Name = Ben
Age = 80
Occupation = Engineer
```

36 | The Filter Function

We can use the *filter()* function as an alternative to the *for loop*. If you want to return items from an iterable that match certain criteria, you can use the Python *filter()* function.

Let's say we have a list of names and we want to return a list of names that are lowercase; here is how you can do it using the *filter()* function.

The first example uses a for loop for comparison purposes.

Example 1: Using a for loop

```
names = ['Derick', 'John', 'moses', 'linda']

for name in names:
    if name.islower():
        lower_case.append(name)
print(lower_case)
Output:
['moses', 'linda']
```

Example 2: Using filter function with lambda function

The *filter()* function is a higher-order function. The filter function takes two arguments: a function and a sequence. It uses the function to filter the sequence. In the code below, the filter function uses the lambda function to check for names in the list that are lowercase.

```
names = ['Derick', 'John', 'moses', 'linda']

lower_case = list(filter(lambda x:x.islower(), names))
print(lower_case)
Output:
['moses', 'linda']
```

Example 3: Using filter function with a function

If we do not want to use a *lambda* function, we can write a function and pass it as an argument to the filter function. See the code below:

```
names = ['Derick', 'John', 'moses', 'linda']

# Creating a function
def lower_names(n:str):
    return n.islower()

# passing the function as argument of the filter function
lower_case = list(filter(lower_names, names))
print(lower_case)
Output:
['moses', 'linda']
```

37 | Dictionary Comprehension

Dictionary comprehension is a one-line code that transforms a dictionary into another dictionary with modified values. It makes your code intuitive and concise. It is similar to list comprehension.

Let's say you want to update the values of the dictionary from integers to floats; you can use dictionary comprehension. Below, *k* accesses the dictionary keys, and *v* accesses the dictionary values.

```
dict1 = {'Grade': 70, 'Weight': 45, 'Width': 89}

# Converting dict values into floats
dict2 = {k: float(v) for (k, v) in dict1.items()}
print(dict2)
Output:
{'Grade': 70.0, 'Weight': 45.0, 'Width': 89.0}
```

Let's say we want to divide all the values in the dictionary by 2; here is how we do it.

```
dict1 = {'Grade': 70, 'Weight': 45, 'Width': 89}

# dividing dict values by 2
dict2 = {k: v/2 for (k, v) in dict1.items()}
print(dict2)
Output:
{'Grade': 35.0, 'Weight': 22.5, 'Width': 44.5}
```

38 | DataFrame from Two Lists

The easiest way to create a DataFrame from two lists is to use the *pandas* module. First install pandas with pip:

pip install pandas

Import pandas, and pass the lists to the DataFrame constructor. Since we have two lists, we have to use the *zip*() function to combine the lists.

Below, we have a list of car brands and a list of car models. We are going to create a DataFrame. The DataFrame will have one column called **Brands,** and another called **Models**, and the **index** will be the numbers in ascending order.

```
import pandas as pd

list1 = ['Tesla', 'Ford', 'Fiat']
models = ['X', 'Focus', 'Doblo']

df = pd.DataFrame(list(zip(list1,models)),
                  index =[1, 2, 3],
                  columns=['Brands','Models'])

print(df)
Output:
Brands     Models
1  Tesla      X
2   Ford   Focus
3   Fiat   Doblo
```

39 | Adding a Column to a DataFrame

Let's continue with the tip from the previous tip (38). One of the most important aspects of pandas DataFrame is that it is very flexible. We can add and remove columns. Let's add a column called Age to the DataFrame. The column will have the ages of the cars.

```
import pandas as pd

list1 = ['Tesla', 'Ford', 'Fiat']
models = ['X', 'Focus', 'Doblo']

df = pd.DataFrame(list(zip(list1,models)),
                  index =[1, 2, 3],
                  columns=['Brands','Models'])
# Adding a column to dataFrame
df['Age'] = [2, 4, 3]
print(df)
```

Output:

```
Brands            Models      Age
1  Tesla             X          2
2   Ford          Focus         4
3   Fiat          Doblo         3
```

Dropping or removing a column

Pandas has a *drop()* method that we can use to remove columns and rows from a DataFrame. Let's say we want to remove the column "Models" from the DataFrame above. You can see from the output that the "Models" column has been removed. The *inplace=True* means we want the change to be made on the original DataFrame. On a DataFrame, axis 1 means columns. So, when we pass 1 to the axis parameter, we are dropping a column.

```
df.drop('Models', inplace=True, axis=1)
print(df)
```

Output:

```
      Brands  Age
1      Tesla   2
2      Ford    4
3      Fiat    3
```

40 | Timer Decorator

Below, I created a timer function that uses the *perf_counter* class of the time module. Notice that the *inner()* function is inside the *timer()* function; this is because we are creating a decorator. Inside the *inner()* function is where the "decorated" function will run. Basically, the "decorated" function is passed as an argument to the decorator. The decorator then runs this function inside the *inner()* function.

The *@timer* right before the *range_tracker* function means that the function is being decorated by another function. **To "decorate" a function is to improve or add extra functionality to that function without changing it**. By using a decorator, we are able to add a timer to the *range_tracker* function. We are using this timer to check how long it takes to create a list from a range.

```
import time

def timer(func):
    def inner():
        start = time.perf_counter()
        func()
        end = time.perf_counter()
        print(f'Run time is {end-start:.2f} secs')
    return inner

@timer
def range_tracker():
    lst = []
    for i in range(10000000):
        lst.append(i**2)

range_tracker()
Output:
Run time is 10.25 secs
```

41 | List Comprehension vs Generators

A generator is similar to a list comprehension, but instead of square brackets, you use parenthesis. Generators yield one item at a time, while list comprehension releases everything at once. Below, I compare list comprehension to a generator in two categories:

1. Speed of execution.
2. Memory allocation.

Conclusion

List comprehension executes much faster but takes up more memory. Generators execute a bit slower, but they take up less memory since they only yield one item at a time.

```
import timeit
import sys

# function to check time execution
def timer(_, code):
    exc_time = timeit.timeit(code, number=1000)
    return f'{_}: execution time is {exc_time:.2f}'

# function to check memory allocation
def memory_size(_, code):
    size = sys.getsizeof(code)
    return f'{_}: allocated memory is {size}'

one = 'Generator'
two = 'list comprehension'

print(timer(one, 'sum((num**2 for num in range(10000)))'))
print(timer(two, 'sum([num**2 for num in range(10000)])'))
print(memory_size(one,(num**2 for num in range(10000))))
print(memory_size(two,[num**2 for num in range(10000)]))
```

```
Output:
Generator: execution time is 5.06
list comprehension: execution time is 4.60
Generator: allocated memory is 112
list comprehension: allocated memory is 85176
```

42 | Writing to File

Let's say you have a list of names and you want to write them in a file, and you want all the names to be written vertically. Here is a sample code that demonstrates how you can do it. The code below creates a CSV file. We tell the code to write each name on a new line by using the escape character ('\n'). Another way you can create a CSV file is by using the CSV module.

```
names = ['John Kelly', 'Moses Nkosi', 'Joseph Marley']

with open('names.csv', 'w') as file:
    for name in names:
        file.write(name)
        file.write('\n')

# reading CSV file
with open ('names.csv', 'r') as file:
    print(file.read())
Output:
John Kelly
Moses Nkosi
Joseph Marley
```

Using CSV module

If you don't want to use this method, you can import the CSV module. CSV is a built-in module that comes with Python, so there is no need to install it. Here is how you can use the CSV module to accomplish the same thing. Example code on the next page:

```
import csv

names = ['John Kelly', 'Moses Nkosi', 'Joseph Marley']

with open('names.csv', 'w') as file:
    for name in names:
        writer = csv.writer(file,lineterminator = '\n')
        writer.writerow([name])

# Reading the file
with open ('names.csv', 'r') as file:
    print(file.read())
```

Output:

```
John Kelly
Moses Nkosi
Joseph Marley
```

43 | Merge PDF Files

If we want to merge PDF files, we can do it with Python. We can use the PyPDF2 module. With this module, you can merge as many files as you want. First, install the module using pip.

pip install pyPDF2

```
import PyPDF2
from PyPDF2 import PdfFileMerger, PdfFileReader,

# Create a list of files to merge
list1 = ['file1.pdf', 'file2.pdf']


merge = PyPDF2.PdfFileMerger(strict=True)
for file in list1:
    merge.append(PdfFileReader(file,'rb+'))

# Merge the files and name the merged file
merge.write('mergedfile.pdf')
merge.close()
# Reading the created file
created_file = PdfFileReader('mergedfile.pdf')
created_file
Output:
<PyPDF2.pdf.PdfFileReader at 0x257d1c8ba90>
```

For this code to run successfully, make sure that the files you are wanting to merge are saved in the same location as your Python file. Create a list of all the files that you are trying to merge. The output simply confirms that the merged file has been created. The name of the file is *mergedfile.pdf*.

44 Return vs Yield

Do you understand the distinction between a return statement and a yield statement in a function? The return statement returns one element and ends the function. The yield statement returns a "package" of elements called a generator. You have to "unpack" the package to get the elements. You can use a for loop or the *next()* function to unpack the generator.

Example 1: Using return statement

```
def num (n: int) -> int:
    for i in range(n):
        return i


print(num(5))
Output:
0
```

You can see from the output that once the function returns **0**, it stops. It does not return all the numbers in the range.

Example 2: Using yield

```
def num (n: int):
    for i in range(n):
        yield i

# creating a generator
gen = num(5)

#unpacking generator
for i in gen:
    print(i, end = '')
Output:
0 1 2 3 4
```

The yield function returns a "package" of all the numbers in the range. We use a *for loop* to access the items in the range.

45 | High Order Functions

A high-order function is a function that takes another function as an argument or returns another function. The code below demonstrates how we can create a function and use it inside a high-order function. We create a function called *sort_names*, and we use it as a key inside the *sorted()* function. By using index[0], we are basically telling the sorted function to sort names by their first name. If we use [1], then the names would be sorted by the last name.

```
def sort_names(x):
    return x[0]

names = [('John','Kelly'),('Chris','Rock'),
         ('Will','Smith')]

sorted_names = sorted(names, key= sort_names)
print(sorted_names)
Output:
[('Chris', 'Rock'), ('John', 'Kelly'), ('Will', 'Smith')]
```

If we don't want to write a function as above, we can also use the *lambda* function. See below:

```
names = [('John','Kelly'),('Chris','Rock'),
         ('Will','Smith')]

sorted_names = sorted(names, key= lambda x: x[0])
print(sorted_names)
Output:
[('Chris', 'Rock'), ('John', 'Kelly'), ('Will', 'Smith')]
```

46 | Grammar Errors

Did you know that you can correct grammar errors in text using Python? You can use an open-source framework called Gramformer. Gramformer (created by Prithviraj Damodaran) is a framework for highlighting and correcting grammar errors in natural-language text. Here is a simple code that demonstrates how you can use gramformer. to correct errors in a text.

First, you need to install it. Run this below:

```
!pip3 install torch==1.8.2+cu111 torchvision==0.9.2+cu111 torchaudio==0.8.2 -
f https://download.pytorch.org/whl/lts/1.8/torch_lts.html


!pip3 install -U git+https://github.com/PrithivirajDamodaran/Gramformer.git
```

```
from gramformer import Gramformer

# instantiate the model
gf = Gramformer(models=1, use_gpu=False)

sentences = [
             'I hates walking night',
             'The city is were I work',
             'I has two children'
              ]

for sentence in sentences:
    correct_sentences = gf.correct(sentence)
    print('[Original Sentence]', sentence)
    for correct_sentence in correct_sentences:
        print('[Corrected sentence]', correct_sentence)
```

```
Output:
[Original Sentence] I hates walking night
[Corrected sentence] I hate walking at night.
[Original Sentence] The city is were I work
[Corrected sentence] The city where I work.
[Original Sentence] I has two children
[Corrected sentence] I have two children.
```

47 | Zen of Python

A Zen of Python is a list of 19 guiding principles for writing beautiful code. Zen of Python was written by Tim Peters and later added to Python.

Here is how you can access the Zen of Python.

```
import this

print(this)
```

```
Output:
```

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

48 | Sorted by Pprint

Did you know that you can print out a sorted dictionary using the pprint module? Below, we use the module to print out a dictionary in ascending order. The dictionary is sorted by key.

```
import pprint

a = {'c': 2, 'b': 3, 'y': 5,'x': 10}

pp = pprint.PrettyPrinter(sort_dicts=True)
pp.pprint(a)
Output:
{'b': 3, 'c': 2, 'x': 10, 'y': 5}
```

Please note that pprint does not change the order of the actual dictionary; it just changes the printout order.

Insert underscore

Pprint can also be used to insert a thousand separator into numbers. Pprint will insert an underscore as a thousand separator. It has a parameter called *underscore_numbers*. All we have to do is set it to True. See the code below:

```
import pprint

arr = [1034567, 1234567, 1246789, 12345679, 987654367]

pp = pprint.PrettyPrinter(underscore_numbers=True)
pp.pprint(arr)
Output:
[1_034_567, 1_234_567, 1_246_789, 12_345_679, 987_654_367]
```

49 | Convert Picture to Grey Scale

Do you want to convert a color image into grayscale? Use Python's cv2 module.

First install cv2 using > **pip install opencv-python**<

Below we are converting a color book image to grayscale. You can replace that with your own image. You must know where the image is stored.

When you want to view an image using CV2, a window will open. The *waitkey* indicates how long we expect the display window to be open. If a key is pressed before the display time is over, the window will be destroyed or closed.

```
import cv2 as cv


img = cv.imread('book.jpg')

# show the original image
img1 = cv.imshow('Original', img)
cv.waitKey(5)

# Converting image to Gray
grayed_img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

# Show grayed-out image
img2 = cv.imshow('grayed_image', grayed_img)
cv.waitKey(5000)

#Save image
cv.imwrite('grayed.jpg', grayed_img)
```

50 | Time it with timeit

If you want to know the execution time of a given code, you can use the *timeit()* module. Below, we create a function called timer that uses *timeit()* from the *timeit* module. We're basically using ***timeit*** to determine how long it takes to run **sum(num**2 for num in range(10000))**.The first parameter of the timeit function in the code below is *stmt*. This is where we pass the Python code that we want to measure. This parameter only takes a string as an argument, so we have to ensure that we convert our code to a string format. The *number* parameter is basically the number of executions we want to run on this code.

```
import timeit

def timer(code):
    tm = timeit.timeit(code,number=1000)
    return f'Execution time is {tm:.2f} secs.'

if __name__ == "__main__":
    print(timer('sum(num**2 for num in range(10000))'))
Output:
Execution time is 5.05 secs.
```

It's not necessary to create a function like we did above. You can also use *timeit* without creating a function. Let's simplify the above code by writing it without a function. Remember that the *stmt* parameter only takes a string as an argument; that is why the *test* variable below, which is the code that we pass to the *timeit()* function, is a string.

```
import timeit

test = "sum(num ** 2 for num in range(10000))"
tm = timeit.timeit(stmt=test,number=1000)
print(f'{tm:.2f} secs')
Output
2.20 secs
```

51 | Shortening URL with Python

Most of us have used programs that generate short URLs. Python has a library called **pyshorteners** used to shorten a URL. First install it using pip:

pip install pyshorteners

Once you install it, you can import it into your script. The function below demonstrates how we can use pyshorteners. We pass a very long URL to the function, and it returns a short URL.

```
import pyshorteners


def shorter_url(s: str):
    # iniatialize the shortener
    pys = pyshorteners.Shortener()
    # Using the tinyurl to shorten
    short_url = pys.tinyurl.short(s)
    return 'Short url is', short_url


print(shorter_url(

"https://www.google.com/search?q=python+documentation&newwind
ow=1&sxsrf=ALiCzsYze-"

"G2AArMZtCrJfyfVcqq6z8Rwg%3A1662044120968&ei=2McQY4PaOp68xc8P
yp-qIA&oq=python+do&gs_lcp="

"Cgdnd3Mtd2l6EAEYATIFCAAQgAQyBQgAEIAEMgUIABCABDIFCAAQgAQyBQgA
EIAEMgUIABCABDIFCAAQg"

"AQyBQgAEIAEMgUIABCABDIFCAAQgAQ6BwgAEEcQsAM6DQguEMcBENEDELADE
EM6BwgAELADEEM6BAgj"

"ECc6BAgAEEM6BAguECdKBAhBGABKBAhGGABQ1xBY70JgxlVoAXABeACAAYgB
iAGWCJIBAzAuOZgBA"
    "KABAcgBCsABAQ&sclient=gws-wiz"))
```

```
Output
('Short url is', 'https://tinyurl.com/2n2zpl8d')
```

52 | The Round Function

How do you easily round off a number in Python? Python has a built-in round function that handles such tasks. The syntax is:

round (number, number of digits)

The first parameter is the number to be rounded, and the second parameter is the number of digits a given number will be rounded to. That is the number of digits after the point. The second parameter is optional. Here is an example:

```
num = 23.4567
print(round(num, 2))
Output:
23.46
```

You can see that we have two digits after the decimal point. If we didn't pass the second parameter (2), the number would be rounded to 23.

To round up or down a number, use the *ceil* and *floor* methods from the math module. The *ceil* rounds up a number to the nearest integer greater than the given number. The *floor* method rounds down a number to the nearest integer that is less than the given number.

```
import math

a = 23.4567
# rounding up
print(math.ceil(a))
# rounding down
print(math.floor(a))
Output:
24
23
```

53 | Convert PDF Files to Doc

Did you know that you can convert a PDF file to a Word document using Python?

Python has a library called pdf2docs. With this library, you can convert a **pdf file** to a **word** document with just a few lines of code. Using pip, install the library;

pip install pdf2docx

We are going to import the Converter class from this module. If the file is in the same directory as your Python script, then you can just provide the name of the file instead of a link. The new doc file will be saved in the same directory.

```
from pdf2docx import Converter

# path to your pdf file
pdf = 'file.pdf'

# Path to where the doc file will be saved
word_file = 'file.docx'

#instantiate converter
cv = Converter(pdf)
cv.convert(word_file)

#close file
cv.close()
```

54 | Text from PDF File

We can use the Python library **PyPDF2** to extract text from PDFs. First, install the library using pip;

pip install PyPDF2

We use the **PdfFilereader** class from the module. This class will return the number of files in the pdf document, and it has a get page method, which we can use to specify the page we want to extract information from. Below, we extract text from the book 50 Days of Python.

```
import PyPDF2

# Open a pdf file
pdf_file = open('50_Days_of_Python.pdf', 'rb')

# Read pdf reader
read_file = PyPDF2.PdfFileReader(pdf_file)

# Read from a specified page
page = read_file.getPage(10)

# extracting text from page
print(page.extractText())
# closing pdf file
pdf_file.close()
```

Output:

```
Day 2 : Strings to Integers
Write a function called convert _add  that takes a list of strings
as an argument and converts it to integers and sums the list. For
example ['1', '3', '5'] should be converted to [1, 3, 5] and
summed to 9.

Extra Challenge: Duplicate Names

Write a function called check _duplicates that takes a list of
strings as an argument. The function should check if the list
has
any duplicates. If there are duplicates, the function should
return
the duplicates. If there are no duplicates, the function should
```

return "No duplicates ". For example, the list of fruits below should return apple as a duplicate and list of names should return "no duplicates ".
fruits = ['apple', 'orange', 'banana', 'apple']
names = ['Yoda ', 'Moses ', 'Joshua ', 'Mark ']

55 | Libraries Locations

Have you ever wondered where your installed libraries are located on your machine? Python has a very simple syntax for getting the locations of installed libraries. If you have installed pandas, here is how to find its location. Here is what I get when I run it on my computer:

```
import pandas

print(pandas)
Output:
<module 'pandas' from 'C:\\Users\\Benjamin\\AppData\\Local\\Programs\\Python\\Python310\\lib\\site-packages\\pandas\\__init__.py'>
```

If a module is built-in (a built-in module comes preinstalled with Python), you will not get information on where it is installed. Let's try to print the sys module.

```
print(sys)
Output:
<module 'sys' (built-in)>
```

56 | Create a Barcode

How about creating a barcode using Python? You can use python-barcode to generate different types of objects. First, install the barcode: **Pip install python-barcode**

Below, we are going to create an ISBN13 barcode. Please note that this module supports many other types of barcodes. We are going to generate a PNG image of the barcode. We will use the pillow module to view the image. You can install pillows by running: **pip install pillow**

Pass a 12-digit number as a string.

```
from barcode import ISBN13
from barcode.writer import ImageWriter
from PIL import Image

num = '979117528719'
# saving image as png
bar_code = ISBN13(num, writer=ImageWriter())
# save image
bar_code.save('bar_code')
#read the image using pillow
img = Image.open("bar_code.png")
img.show()
```

Output

57 | Indices Using Len & Range Functions

If you want to get the indices of items in a sequence, such as a list, we can use the *len()* and *range()* functions if you don't want to use the *enumerate()* function. Let's say we have a list of names and we want to return the indices of all the names in the list. Here is how we can use the *len()* and *range()* functions:

```
names = ['John', 'Art', 'Messi']

for i in range(len(names)):
    print(i, names[i])
```

Output:

```
0 John
1 Art
2 Messi
```

If we want to return just the index of one of the names in the list, we can combine the *len()* and *range()* functions with a conditional statement. Let's say we want the index of the name "Messi"; here is how we can do it:

```
names = ['John', 'Art', 'Messi']


for i in range(len(names)):
    if names[i] == 'Messi':
        print(f'The index of the name {names[i]} is {i}')
        break
```

Output:

```
The index of the name Messi is 2
```

58 | Convert Emoji to Text

Did you know that you can extract text from emojis? Let's say you have a text with emojis and you don't know what the emojis mean. You can use a Python library to convert the emojis to text.

First install the library.

Pip install demoji

The **demoji** library returns a dictionary of all the emojis in a text. The emoji is the *key*, and the value is the emoji converted to text.

```
import demoji

txt = "I spent the day at 🐂. Today its very 🔥🥵. I cant wait for winter 🥶"
demoji.findall(txt)
```

Output

```
{'🔥': 'fire',
 '🥶': 'cold face',
 '🏋': 'person lifting weights',
 '🥵': 'hot face'}
```

59 | Currency Converter

In Python, with just a few lines of code, you can write a program that converts one currency into another using up-to-date exchange rates. First, install the forex library using: **pip install forex-python**

The code below will convert any currency. You just have to know the currency codes of the currencies you are about to work with. Try it out.

```
from forex_python.converter import CurrencyRates

# Instantiate the converter
converter = CurrencyRates()

def currency_converter():
    # Enter the amount to convert
    amount = int(input("Please enter amount to convert: "))
    # currency code of the amount to convert
    from_currency = input("Enter the currency code of "
                          "amount you are converting : ").upper()
    # currency code of the amount to convert to
    to_currency = input("Enter the currency code you "
                        "are converting to: ").upper()
    # convert the amount
    converted_amount = converter.convert(from_currency,
                                         to_currency, amount)
    return f' The amount is {converted_amount:.2f} and ' \
           f'the currency is {to_currency}'

print(currency_converter())
```

60 | Generate Custom Font

Python has a cool library that you can use to generate custom fonts. This library is designed for the purpose of creating fancy texts for our programs. For example, we can use a generated font to create a cool article title. Below, we are going to demonstrate how we can create a fancy title with the module. Install with pip;

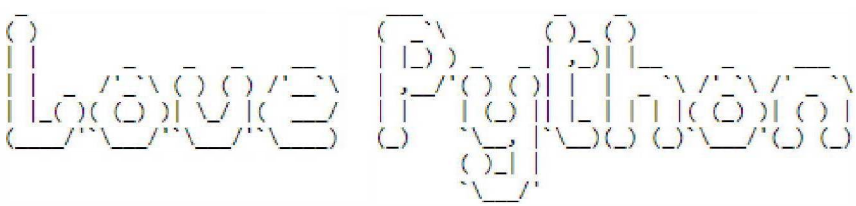
pip install pyfiglet

The *figlet_format()* method takes two arguments: the text we want to format and the font. The font parameter is optional. If we do not pass an argument for the font, the default font will be applied to our text.

```
import pyfiglet

text = pyfiglet.figlet_format("Love Python",
font="puffy")
print(text)
```

Output:



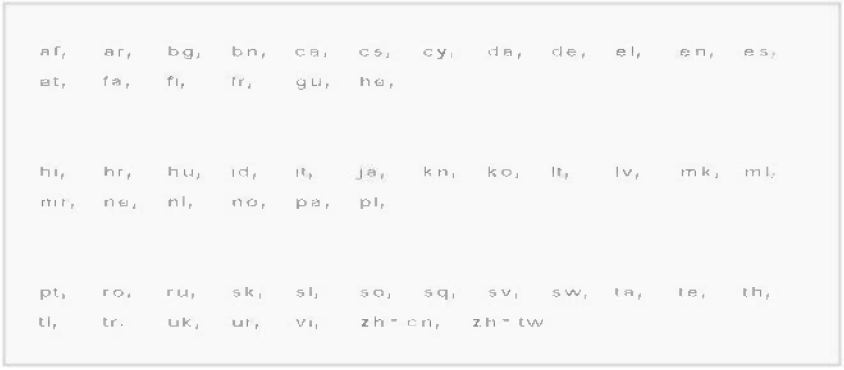
You can also generate a list of fonts that you can use for your text. Run this code below and you will get a list of fonts that are available.

```
import pyfiglet

print(pyfiglet.FigletFont.getFonts())
```

61 | Language Detector

You can detect language in a text using a Python library called **langdetect**. At the moment, this detector supports about 55 languages. Here are the supported languages below:



To use the library, install using pip;

pip install langdetect

It takes a string of words and detects the language. Below, it can be detected that the language is Japanese.

```
from langdetect import detect

# Language to detect
lang = detect("• • • • ")

print(lang)
Output:
ja
```

62 | Refresh URL with Selenium

Did you know that you can refresh a URL using Python? Usually, to refresh a page, we have to do it manually. However, we can automate the process with just a few lines of code. We will use the **selenium** module for this. Install the following:

pip install selenium
pip install webdriver-manager

Now, let's write the code. I am using the Chrome web browser, so I will need Chrome dependencies. If you are using another browser, you will need to install a driver for that browser. So, we need the URL link of the website that we want to open. The time is the number of seconds we want to wait before refreshing the page. The code will automatically refresh the page once the waiting time is over.

```
import time
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from webdriver_manager.chrome import ChromeDriverManager

# Url to open and refresh
url = "url to open and refresh"

# installs web driver for chrome
driver = webdriver.Chrome(service=Service(ChromeDriverManager()
                                          .install()))

driver.get(url)
# waiting time before refresh
time.sleep(10)
driver.refresh()
```

63 | Substring of a String

If you want to check if a string is a substring of another string, you can use the **in** and **not in** operators. Let's say we want to test if "like" is a substring of string ***s***. Here is how we do it using the **in** operator: The **in** operator will evaluate to **True** if "like" is a substring of s and **False** if it is not.

```
s = 'Please find something you like'

if 'like' in s:
    print('Like is a substring of s')
else:
    print('Like is not a substring of s')
Output:
Like is a substring of s
```

We can also use the "**not in**" operator. The "**not in**" operator is the opposite of the in operator.

```
s = 'Please find something you like'

if 'like' not in s:
    print('Like is not a substring of s')
else:
    print('Like is a substring of s')
Output:
Like is a substring of s
```

Python advises only using the **find()** method to know the position of a substring. If we wanted to find the position of "something" in a string, here is how we would use the **find()** method: The find method returns the index where the substring starts.

```
s = 'Please find something you like'

print(s.find('something'))
Output:
12
```

64 | Difference Between Two Lists

If you have two lists and you want to find the difference between the lists, that is, elements that are in **list a** but not in **list b,** use the **set().difference(set())** method.

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

difference = set(a).difference(set(b))
print(list(difference))
Output:
[8, 4, 6]
```

Another naive way would be to use a *for loop*. See below:

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

difference = []
for number in a:
    if number not in b:
        difference.append(number)
print(difference)
Output:
[6, 8, 4]
```

You can also convert the above code into a list comprehension.

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

dif = [number for number in a if number not in b]
print(dif)
Output:
[6, 8, 4]
```

65 | Sorting a List of Dictionaries

If you have a list of dictionaries and you want to sort them by their values, you can use the *itemgetter* class from the operator module and the sorted function. Here is an example to demonstrate.

```
from operator import itemgetter

d = [{"school":"yale", "city": "Beijing"},
     {"school":"cat", "city": "Cairo"}]

sorted_list = sorted(d, key=itemgetter('school'))
print(sorted_list)
Output:
[{'school': 'cat', 'city': 'Cairo'}, {'school': 'yale',
'city': 'Beijing'}]
```

The above example sorts a list in **ascending order**. If you wanted to sort the list in **descending order**, you would have to set the reverse parameter in the sorted function to True. You can see below that the list has been sorted in descending order.

```
from operator import itemgetter

d = [{"school":"yale", "city": "Beijing"},
     {"school":"Cat", "city": "Cairo"}]

sorted_list = sorted(d, key=itemgetter('school'),reverse=True)
print(sorted_list)
Output:
[{'school': 'yale', 'city': 'Beijing'}, {'school': 'Cat',
'city': 'Cairo'}]
```

66 | Bytes to String

There are two methods we can use to convert bytes into strings. Method number one is to use the *str()* function. The second method is to use the decode method. First, here is a byte data type:

```
s = b'Love for life'
print(type(s))
Output:
<class 'bytes'>
```

Now let's convert this into a string.

Method 1: Using the str function

```
s = b'Love for life'
str1 = str(s,"UTF-8")
print(type(str1))
print(str1)
Output:
<class 'str'>
Love for life
```

Method 2: Using the decode method

```
s = b'Love for life'

str1 = s.decode()
print(type(str1))
print(str1)
Output:
<class 'str'>
Love for life
```

67 | Multiple Input from User

What if you want to get multiple inputs from a user? How do you do it in Python? Python uses the *input()* function to get input from the user. The *input()* function takes one input at a time; that's the default setting. How can we modify the input function so it can accept multiple inputs at a time? We can use the *input()* function together with the string method, *split()*. Here is the syntax below:

input().split()

With the help of the *split()* function, the *input()* function can take multiple inputs at the same time. The *split()* method separates the inputs. By default, the split method separates inputs right at the whitespace. However, you can also specify a separator. Below, we ask a user to input three (3) numbers. We use the *split()* method to specify the separator. In this case, we separate the inputs with a comma (,). Then we calculate the average of the three numbers. If the inputs are 12, 13, and 14, we get the output below:

```
a, b, c = input("Please input 3 numbers: ").split(sep=',')

average = (int(a) + int(b) + int(c))/3
print(average)
Output:
13
```

We can also get multiple values from a user. Below, we ask a user to input multiple names using list comprehension. If the user enters Kelly, John, Trey, and Steven, the output will be:

```
# Getting multiple values using list comprehension
n = [nam for nam in input("Enter multiple names: ").split(sep=',')]

print(n)
Output:
['Kelly', ' John', ' Trey', 'Steven']
```

68 | The _iter_() Function

If you want to create an iterator for an object, use the *iter()* function. Once the iterator object is created, the elements can be accessed one at a time. To access the elements in the iterator, you will have to use the *next()* function. Here is an example to demonstrate this:

```
names = ['jake', "Mpho", 'Peter']
# Creating an iterator
iter_object = iter(names)
# Accessing items using next function
name1 = next(iter_object)
print('First name is', name1)
name2 = next(iter_object)
print('Second name is', name2)
name3 = next(iter_object)
print('Third name is', name3)
```

Output:

```
First name is jake
Second name is Mpho
Third name is Peter
```

Because the list only has three elements, attempting to print the fourth element in the iterable will result in a *StopIteration* error.

We can also elegantly put the *next()* function in a loop. See below:

```
names = ['Jake', "Mpho", 'Peter']
iter_object = iter(names)
while True:
    # accessing the items in object using next func
    try:
        print(next(iter_object))
    except:
        break
```

Output:

```
Jake
Mpho
Peter
```

Why use iterators? Iterators have better memory efficiency.

69 | Two Lists into a Dict

Sometimes, while working with lists, you may want to change the data type to a dictionary. That is, you may want to combine the lists into a dictionary. To do this, you may have to use the *zip()* function and the *dict()* function. The *zip()* function takes two iterables and pairs the elements. The first element in iterable one is paired with the first element in iterable two, and the second element with another second element, etc. The *zip()* function returns an iterator of tuples. The dict() function will convert the paired elements into a key-value combination, creating a dictionary.

```
list1 = ['name', 'age', 'country']
list2 = ['Yoko', 60, 'Angola']

dict1 = dict(zip(list1,list2))
print(dict1)
```

Output:

```
{'name': 'Yoko', 'age': 60, 'country': 'Angola'}
```

If an element in the list cannot be paired with another element, then it will be left out. Let's say **list1** has four elements and **list2** has five; the fifth item in list2 will be left out. You can see below that "Luanda" has been left out.

```
list1 = ['name', 'age', 'country']
list2 = ['Yoko', 60, 'Angola', "Luanda"]

dict1 = dict(zip(list1,list2))
print(dict1)
Output:
{'name': 'Yoko', 'age': 60, 'country': 'Angola'}
```

70 | Finding Permutations of a string

Permutations of a string are different orders in which we can arrange the elements of the string. For example, given an "ABC" string, we can rearrange it into ["ABC," "ACB," "BAC," "BCA," "CAB," "CBA."]. In Python, the easiest way to find permutations of a string is to use *itertools*. Itertools has a permutation class. Here is how we can do it, using itertools.

```
from itertools import permutations

def get_permutations(s: str):
    arr = []
    for i in permutations(s):
        arr.append(''.join(i))
    return a


print(get_permutations('ABC'))
```

Output:

```
['ABC', 'ACB', 'BAC', 'BCA', 'CAB', 'CBA']
```

Another way to do it.

```
def find_permute(string, j ):
    if len(string) == 0:
        print(j, end="  ")


    for i in range(len(string)):
        char = string[i]
        s = string[0:i] + string[i + 1:]
        find_permute(s, j + char)
    return j

print(find_permute('ABC', ''))
```

Output:

```
ABC ACB BAC BCA CAB CBA
```

71 | Unpacking a List

Sometimes you want to unpack a list and assign the elements to different variables. Python has a method that you can use to make your life easier. We can use the Python unpacking operator, asterisk (*). Below is a list of names. We want to get the boy's name and assign it to the variable "boy_name." The rest of the names will be assigned to the variable "girls_name." So, we assign (unpack from the list) the first item on the list to the boy variable. We then add "*" to the **girls_name** variable. By adding * to the girl's name, we are basically telling Python that once it unpacks the male name, all the other remaining names must be assigned to the **girls_name** variable. See the output below:

```
names = [ 'John', 'Mary', 'Lisa', 'Rose']

boy_name, *girls_name = names
print(boy)
print(girls)
Output:
John
['Mary', 'Lisa', 'Rose']
```

If the name "John" was at the end of the list, we would put the name with the asterisk at the beginning. See below:

```
names = [ 'Rose', 'Mary', 'Lisa', 'John']

*girls, boy = names
print(boy)
print(girls)
Output:
John
['Mary', 'Lisa', 'Rose']
```

72 | Type Hints

Python provides a way to give hints on what type of argument is expected in a function or what data type a function should return. These are called "type hints." Here is an example of how typing hints are used.

```
def translate(s: str) -> bool:
    if s:
        return True
```

Here we have a simple function. Notice that the function has one parameter, **s**. The "str" after the s parameter is a typing hint. It simply means that the function expects (or hints) that the argument that should be passed for parameter "s" must be a string. Notice the "-> bool" at the end? This is another typing hint. It hints that it expects the return value of the function to be a Boolean value. Now, hints are not enforced at runtime by Python; that means that if a non-string argument is passed and a non-Boolean value is returned by the function, the code will run just fine.

Why type hints? Type hints simply help to make the code more readable and more descriptive. By just looking at the hint annotations, a developer will know what is expected of the function.

73 | File Location

Do you want to know the directory of your Python file? Use the **os** module. The **os** module has a **getcwd()** method that returns the location of the directory. Simply type:

```
import os

directory_path = os.getcwd()
print(directory_path)
```

This code will return the location of the current working directory.

74 | Python Deque

If you want to add and pop elements from both sides of a list, you can use deque (double-ended queue). Unlike a normal list that lets you add and remove elements from one end, deque allows you to add and remove elements from both the left and right sides of the list. This makes it quite handy if one has a big stack. Deque is found in the *collections* module. In the example below, see how we are able to append elements at both ends of the list.

```
from collections import deque

arr = deque([1, 3])
# appending on the left end of the list
arr.appendleft(5)
# appending on the right end of the list
arr.append(7)
print(arr)
Output:
 deque([5, 1, 3, 7])
```

Deque also makes it easy to pop (remove) elements on both ends of the list. See the example below:

```
from collections import deque

arr = deque([1, 3, 9, 6])
# pop element on the left end of the list
arr.popleft() # pops 6
# pop element on the right end of the list
arr.pop() # pops 1
print(arr)
Output:
deque([3, 9])
```

You can see from the output that we have popped elements from both ends of the list.

75 | Python ChainMap

What if you have a number of dictionaries and you want to wrap them into one unit? What do you use? Python has a class called chainMap that wraps different dictionaries into a single unit. It groups multiple dictionaries together to create one unit. ChainMap is from the *collections* module. Here is how it works:

```
from collections import ChainMap

x = {'name': 'Jon','sex': 'male'}
y = {'name': 'sasha', 'sex': 'female'}

dict1 = ChainMap(x, y)

print(dict1)
```

```
Output:
ChainMap({'name': 'Jon', 'sex': 'male'}, {'name':
'sasha', 'sex': 'female'})
```

We can also access the keys and values of the dictionaries wrapped in a ChainMap. Below, we print out the keys and values of the dictionaries.

```
from collections import ChainMap

x = {'name': 'Jon','sex': 'male'}
y = {'car': 'bmw', 'make': 'x6'}

dict1 = ChainMap(x, y)

print(list(dict1.keys()))
print(list(dict1.values()))
```

```
Output:
['car', 'make', 'name', 'sex']
['bmw', 'x6', 'Jon', 'male']
```

76 | Progress Bar with Python

When you are executing loops (especially really large ones), you can show a smart progress bar. A library called **tqdm** creates a progress meter that keeps track of the progress of your loop. To install the library, run:

Pip install tqdm

Let's say you have a range of 100000 that you want your loop to run through, and after every loop, you want your code to sleep for 0.001 sec. Here's how to do it with **tqdm** so you can monitor the loop's progress. When you run this code, you will get the progress meter below as the output.

```
from tqdm import tqdm

for i in tqdm(range(100000)):
    pass
    time.sleep(0.001)
```

```
Output:
   9%|▉         | 8503/100000 [02:12<27:29, 55.48it/s]
```

You can also put a description in the function to make the progress meter more intuitive.

```
for i in tqdm(range(100000), desc='progress'):
    pass
    time.sleep(0.001)
```

```
Output
progress:   4%|▍         | 4197/100000 [01:05<24:41, 64.69it/s]
```

You can now see the word "progress" in the progress meter. You can put in whatever description you want. Explore the module some more.

77 | Convert Text to Handwriting

You can convert text into handwriting with Python. Python has a module called **pywhatkit**. Install pywhatkit using pip.

pip install pywhatkit

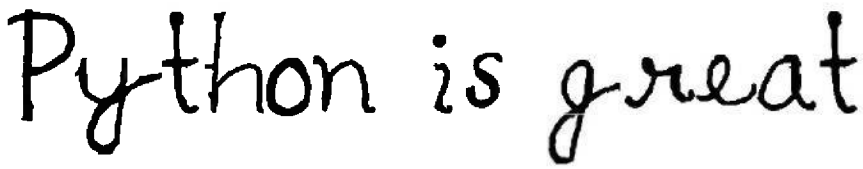
Below, we use *pywhatkit* to convert text to handwriting and save it as an image. We then use the cv2 library to view the image. Run this code below:

```
import pywhatkit
import cv2 as cv


# text to convert to handwriting
text = 'Python is great'
# converting text to handwriting and saving as image
pywhatkit.text_to_handwriting(text,
save_to='new_text.png')

# reaading image using cv
hand_text = cv.imread("new_text.png")
cv.imshow("hand_text", hand_text)
cv.waitKey(0)
cv.destroyAllWindows()
```

Output:



78 | Taking a Screenshot

You can take a screenshot with Python. Python has a library called **pyautogui**, which is an automation tool. Install the library using pip:

pip install pyautogui

Below, we use pyautogui to take a screenshot. We then save the image and convert it from RGB to BGR using cv2 and numpy. We convert the image so it can be read by the cv2 library. Run this code below to see the magic. You can install the other libraries using:

Pip install numpy

pip install opencv-python

```
import pyautogui
import numpy as np
import cv2 as cv

# Taking screenshot
image = pyautogui.screenshot()

image.save('my_screenshot.png')
# convert RGB 2 BGR
image = cv.cvtColor(np.array(image),
                    cv.COLOR_RGB2BGR)

cv.imshow("image", image)
cv.waitKey(0)
cv.destroyAllWindows()
```

79 | Return Multiple Function Values

By default, a function returns one value, and it stops. Below, we have 3 values in the *return* statement. When we call the function, it returns a tuple of all three items.

```
def values():
    return 1, 2, 3

print(values())
```

```
Output:
(1, 2, 3)
```

What if we want to return three separate values? How do we do it in Python? We create a variable for each value we want to return. This makes it possible to access each value separately. See the code below:

```
def values():
    return 1, 2, 3


x, y, z = values()

print(x)
print(y)
print(z)
```

```
Output:
1
2
3
```

80 | Download YouTube Videos

Python makes it super easy to download YouTube videos. With just a few lines of code, you will have your favorite videos saved on your computer. The Python library that you need to download videos is **pytube**. Install it with:

pip install pytube

First, we import the module into our script. We then get the link to the video we are trying to download.

```
from pytube import YouTube
yt_video = YouTube ("<video link>")
```

Next, we set the type of file or extension that we want to download and the resolution of the video.

```
v_file = yt_video.streams.filter(file_extension="mp4")\
       .get_by_resolution("360p")
```

We then download the file. You can also input the path where you want the file to be saved.

```
V_file.download("path to save file.")
```

81 | Convert a String to a List

In Python, we can easily convert a string to a list of strings. We can combine the *list()* function with the *map()* function. The map function returns a map object, which is an iterator. The *map()* function takes two arguments: a function and an iterable. Below, the list is an iterable, and *str()* is the function. We then use the *list()* function to return a list. The *split()* method divides or splits a string at whitespaces.

```
s = "I love Python"

str1 = list(map(str,s.split()))
print(str1)
Output:
['I', 'love', 'Python']
```

You can convert a list of strings back to a string using the *map()* function in conjunction with the *join()* method. Let's convert the output of the above code back to a string using the *map()* and *join()* methods.

```
list1 = ['I','love','Python']

str1 = ' '.join(map(str, list1))
print(str1)
Output:
I love Python
```

82 | Loop Over Multiple Sequences

What if you have two sequences and you want to loop over them at the same time? How can you go about it in Python? This is when the *zip()* function comes in handy. The *zip()* function creates pairs of items in the sequences. The element at index one in sequence one is paired with the element at index one in sequence two. This process repeats for all the other indexes. Let's demonstrate how we can loop over two sequences at the same time.

```
# list one
first_names = ['George','Keith', 'Art']
# list two
last_names = ['Luke', 'Sweat','Funnel']

for first, last in zip(first_names,last_names):
    print(first, last)
```

```
Output:
George Luke
Keith Sweat
Art Funnel
```

You can see from the output that we have combined the two lists using the *zip()* function. We have paired the first names with the last names.

83 | Extend vs. Append

Extend and append are both list methods. They both add elements to an existing list. The *append()* method adds one (1) item to the end of the list, whereas the *append()* method does not. The *extend()* method adds multiple items to the end of the list. Below, we use the *append()* method to append *numbers2* to *numbers1*. You can see that the whole list of *numbers2* has been appended to numbers1 as one (1) item.

```
numbers1 = [1, 2, 3]
numbers2 = [4, 5, 6]

# using append method
numbers1.append(numbers2)
print(numbers1)
Output:
[1, 2, 3, [4, 5, 6]]
```

When we use the *extend()* method, notice the difference from the *append()* method above. The *extend()* method takes the elements in *number2*, and appends them to *numbers1*, one item at a time, not as a whole list. See below:

```
numbers1 = [1, 2, 3]
numbers2 = [4, 5, 6]

# using extend method
numbers1.extend(numbers2)
print(numbers1)
Output:
[1, 2, 3, 4, 5, 6]
```

84 | Memory and _slots_

What are __slots__? Slots are used in classes. They are basically attributes that an instance object will have. Here is an example of slots in action:

```
class Cars:
    __slots__ = ["make", 'brand']
```

We expect every object in the Cars class to have these attributes – make and brand.

So why use slots? We use slots because they help save memory space. Class objects take up less space when we use slots. See the difference in object size in the following examples:

```
import sys

class Cars:
    def __init__(self, make, brand):
        self.make = make
        self.brand = brand

print(f'The memory size is {sys.getsizeof(Cars)}')
Output:
The memory size is 1072
```

Example 2

```
class Cars:
    __slots__ = ["make", 'brand']

    def __init__(self, make, brand):
        self.make = make
        self.brand = brand

print(f'The memory size is {sys.getsizeof(Cars)}')
Output:
The memory size is 904
```

You can see from the outputs above that using slots saves memory space.

85 | Watermark Image with Python

With just a few lines of code, we can watermark our images using Python. We are going to use the pillow library for this task. Install the pillow module using:

Pip install pillow

From this library, we are going to import **ImageDraw**, **ImageFont,** and **Image**.

Below is the image we are going to use for demonstration from https://www.worldwildlife.org/

Let's now import the classes from the pillow module and write the code that we need to watermark this image.

```
from PIL import ImageDraw
from PIL import Image
from PIL import ImageFont


 # Pass link to your image location
pic = Image.open('lion.jpeg')
# make a copy of the pic
drawing = ImageDraw.Draw(pic)
# fill color for the font
fill_color =(255,250,250)
# watermark font
font = ImageFont.truetype("arial.ttf", 60)
# Watermark position
position = (0, 0)
# Writing text to picture
drawing.text(position, text='Lion is King',
fill=fill_color, font=font)
pic.show()
# saving image
pic.save('watermarkedimg.jpg')
```

After running this code, we get a watermarked image below:

86 | Extracting Zip Files

Python has a module that you can use to extract zip files. This is an in-built module, so we do not have to install it. If the file is in the same directory as the Python script, we just pass the file name to the code. We use the *with* statement to open the file. This will ensure that the file is automatically closed at the end of the operation. We open the file in read mode because we just want to read the contents of the file. The *printdir()* method returns all the content of the file we are extracting. The *extractall()* method extracts all the files in the zip file.

```
from zipfile import ZipFile


open file in read mode
with ZipFile('file.zip', 'r') as zipFile:  #
    print('printing the contents of the zip file')
    zipFile.printdir()
    # Extracting zip files
    zipFile.extractall()
```

Use this simple code to extract your zip files.

87 | Generate Dummy Data

You can generate fake data using Python. Python has a library called Faker that generates different types of data. Install Faker by:

pip install Faker

Let's say we want to generate a random list of Ten countries, here is how we do it:

```
from faker import Faker

fake = Faker()

for i in range(20):
    print(fake.country())
```

```
Output:
Cameroon
Tajikistan
Ethiopia
Liechtenstein
Netherlands
Grenada
Switzerland
Singapore
Estonia
San Marino
Malaysia
New Zealand
Azerbaijan
Monaco
British Indian Ocean Territory (Chagos Archipelago)
Brazil
Austria
Saint Barthelemy
Zimbabwe
Korea
```

Since this is random, every time you run it, it will generate a different list of countries.

You can also generate profile data using this model. Below, we generate profile data and pass it to pandas to generate a Pandas DataFrame.

```
from faker import Faker
import pandas as pd

fake = Faker()

profiles = [fake.simple_profile() for i in range(10)]
#Generate a dataframe using pandas
df = pd.DataFrame(profiles)
print(df)
```

Output

	username	name	sex	address	mail	birthdate
0	jhanson	Steven Kennedy	M	8563 Jackson Crest Suite 324\nLawrenceland, CA...	cturner@hotmail.com	1964-08-17
1	akane	Dana Martin	F	025 Love Manor\nMeganstad, CA 83700	longbianca@gmail.com	1961-11-07
2	vmcbride	Ariel Greene	F	3959 Cynthia Rapids Suite 794\nWest Devinberg ...	youngjoel@hotmail.com	1912-03-28
3	stephanienguyen	Rebecca Young	F	71951 Zimmerman Overpass\nWest Sonya, FL 82366	benjaminwillis@gmail.com	1945-11-09
4	frostrichard	Suzanne Walker	F	073 Lopez Neck Suite 940\nPort Sarahshire, NY ...	joseboyer@hotmail.com	1975-07-14
5	robert25	Anthony James	M	2831 Roth Motorway\nNew Debra, OH 05797	gbell@yahoo.com	1907-09-04
6	dennis64	Crystal Stevenson	F	344 Gray Loop Apt. 303\nLake Lindsay, IN 22071	douglasbarbara@yahoo.com	1959-11-29
7	chentimothy	Vicki Hanson	F	6751 Rios Ways Apt. 016\nBrownville, NH 66161	hendersonjames@gmail.com	1932-03-16
8	lrodriguez	Eric Rodriguez	M	3267 John Loop Suite 346\nLake Sandraport, AL ...	jacob48@yahoo.com	1964-06-11
9	amandakim	Paul Tanner	M	337 Le Ferry Suite 142\nLake Carlahaven, MT 46444	frenchmanuel@yahoo.com	1996-08-28

There are many other types of data that you can generate with this module. Explore the module further.

88 | Flatten a list with more_itertools

There are many ways to flatten a list. We have covered other methods already. There is another way we can flatten a nested list. We can use a module called **more_itertools**. We will use the *collapse()* class from this module. This is a one-line solution. First, install the module using:

pip install more_itertools

Let's use to flatten a list.

```
import more_itertools

nested_list = [[12, 14, 34, 56], [23, 56, 78]]

print(list(more_itertools.collapse(nested_list)))
Output:
[12, 14, 34, 56, 23, 56, 78]
```

Collapse a list of tuples

The power of the *collapse()* method is that it will flatten any list, even a list of tuples. See below:

```
import more_itertools

list_tuples = [(12, 14, 34, 56),(23, 56, 78),(12, 23, 56)]

print(list(more_itertools.collapse(list_tuples)))
Output:
[12, 14, 34, 56, 23, 56, 78, 12, 23, 56]
```

89 | Factorial of a Number

Below, we write a code that calculates the factorial of a given number. A factorial of a number is the product of all integers from 1 to that number. The factorial of 3 is calculated as (1 * 2 * 3), which is 6. The *for loop* in the function below executes this calculation. Below, we calculate the factorial of the integer 8.

```
def factorial_number(n: int):
    f = 1
    if n < 0:
        return 'Negatives numbers have no factorial'
    else:
        for k in range(1, n + 1):
            f = f * k
        return f'The factorial of number is {f}'


print(factorial_number(8))
Output:
 The factorial of number is 40320
```

90 | List of Prime Numbers

Given a number, can you write a code that returns a list of all the prime numbers in its range? For example, 6 should return [2, 3, 5]. The code below returns a list of all prime numbers in a given range of a given number. A prime number has two factors: one and itself. Prime numbers start at 2. Here is the code below:

```
def prime_numbers() -> list:
    # Empty list to append prime numbers
    prime_num = []

    n = int(input('Please enter a number (integer): '))
    for i in range(0, n + 1):
      # prime numbers start from 2
        if i > 1:
      # A number divisible by any num
      # in the range then not prime number
            for j in range(2, i):
                if i % j == 0:
                    break
            else:
                prime_num.append(i)

    return prime_num

print(prime_numbers())
```

91 | RAM & CPU Usage

Do you want to know the resources your machine is using? Python has a library called **psutil** that calculates the resource usage of a computer. You can find out how much RAM and CPU your computer is using with just a few lines of code. Install the library using pip:

pip install psutil

Below is a script that calculates how much RAM a machine is using. This is what I get when I run this on the machine I am using right now. Since **psutil** gives results in bytes, we use (1024**3) to convert them to gigabytes.

```
import psutil

memory = psutil.virtual_memory()

def ram_usage():
    print(f'Total available memory in gigabytes '
          f'{memory.total/(1024**3):.3f}')
    print(f'Total used memory in gigabytes '
          f'{memory.used/(1024**3):.3f}')
    print(f'Percentage of memory under use:'
          f' {memory.percent}%')

ram_usage()
```

```
Output:
Total available memory in gigabytes 7.889
Total used memory in gigabytes 6.960
Percentage of memory under use: 88.2%
```

If we want to know the CPU usage of the machine, we can use **psutil.cpu_percent**. This will return a usage percentage for each CPU on the machine. If I run this on the machine I am using now, here is what I get:

```
import psutil

cpu_core = psutil.cpu_percent(percpu=True)

for cpu, usage in enumerate(cpu_core):
    print(f"# {cpu+1} CPU: {usage}%")
```

Output:

```
# 1 CPU: 8.8%
# 2 CPU: 6.2%
# 3 CPU: 7.4%
# 4 CPU: 6.2%
```

Run this on your machine to see what you get.

92 | Concatenation vs. Join

There are two ways you can join strings in Python. The first way is by using concatenation (basically joining strings using the + operator). The second way is to use the string method, *join()*. Let's find out which one is faster. We are going to use *perf_counter* from the time module to time the runtime of our codes.

```
a = 'Hello'
b = 'Python'
c = 'world'
f = 'People'
# Using concatenation
start = time.perf_counter()
for _ in range(10000):
    d = a + b + c + f
end = time.perf_counter()
print(f'concatenation time is {end - start:.5f}')
Output:
concatenation time is 0.00311
```

Using the Join method

```
a = 'Hello'
b = 'Python'
c = 'world'
f = 'People'
# using the join method
start = time.perf_counter()
for _ in range(10000):
    d = ''.join([a, b, c, f])
end = time.perf_counter()
print(f'joining time is {end - start:.5f}')
Output:
joining time is 0.00290
```

You can see from the outputs that the *join()* method is faster than concatenation. If you want faster code, use the *join()* method. However, you should note that the *join()* method executes faster when you have more strings to join (more than 3 strings, at least). The concatenation method executes slower

because for every concatenation, Python has to create a string and allocate memory. This slows down the process. But by using the *join()* method, Python only creates one string. This makes the process much faster. Try running the code and documenting your findings. Note that your execution times will be different from my output.

93 | Recursion Limit

Do you want to know the recursion limit in Python? You can use the **sys** module. Here is the code below:

```
import sys

print(sys.getrecursionlimit())
Output:
1000
```

1000 is the default recursion limit in Python. Python sets this limit to prevent stack overflows. However, Python also provides a method to change the recursion limit to suit your needs. Here is how you can adjust the limit.

```
import sys

sys.setrecursionlimit(1200)
print(sys.getrecursionlimit())
Output:
1200
```

You can see that the limit has changed from 1000 to 1200. So, if you ever get a maximum recursion depth error, just make the adjustment.

94 | Country Info Using Python

You can get information about a country using Python. Python has a module called **countryinfo** that returns data about countries. All you have to do is pass the country name to the module. Let's say we want to know about the provinces in Zambia. We enter the country "Zambia" as the argument, and then use the provinces method to return all the provinces in Zambia.

```
from countryinfo import CountryInfo

country = CountryInfo("Zambia")
print(country.provinces())
Output:
['Central', 'Copperbelt', 'Eastern', 'Luapula',
    'Lusaka', 'North-Western', 'Northern', 'Southern',
    'Western']
```

Let's say we want to get information about the currency used by a given country; we just use the currency method. The **currencies()** method is used below to determine China's currency.

```
from countryinfo import CountryInfo

country = CountryInfo("China")
print(country.currencies())
Output:
['CNY']
```

95 | Factorial Using One Line

We saw earlier how we can calculate the factorial of a number using a function. Do you know that you can write a one-line function that calculates the factorial of a number? To make this work, we are going to need the reduce function from *itertools* and the *lambda* function.

```
from functools import reduce

num = 7

f = reduce(lambda a, b: a * b, range(1, num+1))
print(f)
Output:
5040
```

You can change the *num* variable to any integer number and it will still spit out a factorial of that number. Only one line of code—how cool is that? Let's try 10.

```
from functools import reduce

num = 10

f = reduce(lambda a, b: a * b, range(1, num+1))
print(f)
Output:
3628800
```

96 | Spelling Errors

Did you know that you can correct spelling errors in text using Python? Python has a library called *textblob* that corrects spelling errors. Using pip, install the library:

pip install -U textblob

Below, we pass a string to *textblob* that has spelling errors. You can see that the errors have been replaced with the correct spelling. Textblob will pick the most likely correct word for the misspelled word.

```
from textblob import TextBlob


def correct_text(n : str):
    textblob = TextBlob(n)
    corrected_text = textblob.correct()
    return corrected_text

print(correct_text("I make manys speling mistakess."))
Output:
I make many spelling mistakes.
```

Another interesting thing about this library is that you can use it to pluralize words. See below:

```
from textblob import Word

word = Word('Buy potatoe').pluralize()
print(word)
Output:
Buy potatoes
```

97 | List Elements Identical?

Given a list, how do you check if all the elements in the list are identical. We can use the Python *all()* function. This function takes an iterable as an argument and returns True if all the elements are true. We can use it to check if all elements in a list are identical. If they are, it will return True.

```
def check_list(arr: list):
    return all(arr[0] == arr[i] for i in arr)

print(check_list([1,1,1]))
```

```
Output:
True
```

So, the code above returns True because all the elements in the list [1, 1, 1] are identical. What if the elements are not identical? Let's see what happens.

```
def check_list(arr: list):
    return all(arr[0] == arr[i] for i in arr)

print(check_list([1,1,2,3]))
```

```
Output:
False
```

The function returns False because elements in [1, 1, 2, 3] are not identical.

98 | Censor Profanity with Python

If you have bad words in your text that you want to censor, you can use the Python profanity module. Install using pip:

pip install better_profanity

First thing, import profanity from *better_profanity*. Below, we have a string that has a bad word; we are going to pass the string to the module to have the bad word censored.

```
from better_profanity import profanity

text = ' I hate this shit'
censored_text = profanity.censor(text)
print(censored_text)
```

Output:

```
I hate this ****
```

By default, it uses the asterisk (*) to censor bad words. However, you can also set your own censor character. See the example below. We set $ as the set character.

```
text = 'I hate this shit so much'

# Setting the censor character.
censored_text = profanity.censor(text, censor_char='$')
print(censored_text)
```

```
Output:
I hate this $$$$ so much
```

If you just want to check if a string contains a bad word, you can use the *contains_profanity()* method, which will return a Boolean value of True if it contains profanity and False if it has no profanity.

```
text = 'I hate this shit so much'

# Checking if text contains profanity
print(profanity.contains_profanity(text))
```

```
Output:
True
```

You can also create a custom list of words you want to censor. If a text contains a word from your list, then it will be censored.

```
bad_words = [ 'Hate', 'War', 'evil']

# Create a custom list of words to censor
profanity.load_censor_words(bad_words)

text = 'People that love war are evil'

censored_text = profanity.censor(text)
print(censored_text)
Output:
People that love **** are ****
```

99 | Monotonic or Not?

How do you check if an array is monotonic? An array is monotonic if the elements are sorted either in descending order or ascending order. For example, [4, 5, 5, 7] is monotonic. [4, 10, 2, 5] is not monotonic. We use the *all()* function to check if the numbers in the array are ascending or descending. If they are ascending or descending, it will return True. If they are not, it will return False.

```
def check_monotonic(arr: list):
    if all(arr[i] >= arr[i + 1] for i in range(len(arr)-1)):
        return True
    elif all(arr[i] <= arr[i + 1] for i in range(len(arr)-1)):
        return True
    else:
        return False


list1 = [4, 5, 5, 7]
print(check_monotonic(list1))
```

```
Output:
True
```

The code returns True because [4, 5, 5, 7] is monotonic. If we try [4, 10, 2, 5], here is what we get:

```
def check_monotonic(arr: list):
    if all(arr[i] >= arr[i + 1] for i in range(len(arr)-1)):
        return True
    elif all(arr[i] <= arr[i + 1] for i in range(len(arr)-1)):
        return True
    else:
        return False


list1 = [4, 10, 2, 5]
print(check_monotonic(list1))
```

```
Output:
False
```

100 | Find Factors of a Number

Factors of a number are all the numbers that can be divided into a given number without leaving a remainder. We are going to use the range function to get all the numbers to divide the given number with. If any of those numbers does not return a remainder, we want to print out those numbers. Here is the full code, using the *for* loop and the *if* statement.

```
def number_factors(n: int):
    for i in range(1, n + 1):
        if n % i == 0:
            print(i)

print(number_factors(8))
```

```
Output:
1
2
4
8
```

We can also use list comprehension. If the user enters 8, this will be the output:

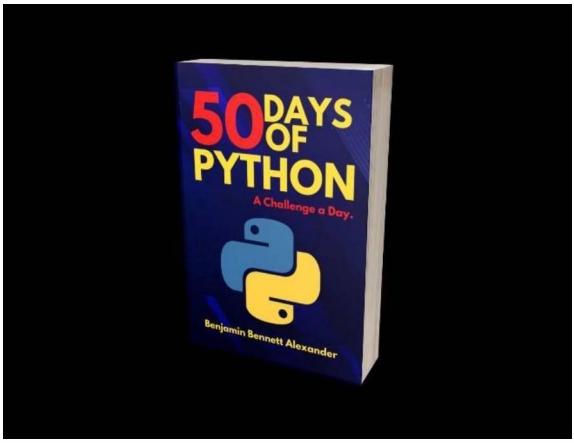
```
n = int(input('Please enter number: '))

factors = [i for i in range(1, n + 1) if n % i == 0]
print(factors)
```

```
Output:
[1, 2, 4, 8]
```

Other Books by the Author

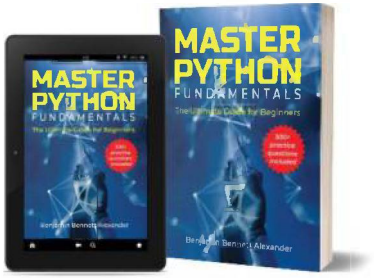
1. **50 Days of Python: A Challenge a Day**



Gumroad link: https://benjaminb.gumroad.com/l/zybjn

Amazon link: https://www.amazon.com/dp/B09TQ83JQB

2. **Master Python Fundamentals: The Ultimate Guide for Beginners**



Amazon link: **https://www.amazon.com/dp/B0BKVCM6DR**

Cover image

Image by liuzishan of Freepik

The End