Essentials For Python Programming

STATS_SOLUTIONS@ryannthegeek

2023-05-11

Contents

List of Figures

List of Tables

Preface

This is the first edition of *Essentials For Python Programming* by Chege G.B as **STATS_SOLUTIONS**@ryannthegeek.

Chapter 1

Numbers and Operators

1. Integers `int` eg 1
2. Floating point numbers (`float`) eg 0.5
3. Complex numbers (`complex`)

1.1 Arithmetic Operators

- ✓ Addition `+`
- ✓ Subtraction `-`
- ✓ Multiplication `*`
- ✓ Division `/` General division.
- ✓ Modulus `%` mode operator, gives the remainder after division
- ✓ Floor Division `//` Gives the whole number after division
- ✓ Exponentiation `**` Is the power of

1.2 Comparison Operators

Comparison operators returns true of false depending on the condition.

- ✓ Greater than `>`
- ✓ Less than `<`
- ✓ Logically equivalent to `==`
- ✓ Not equal to `!=`
- ✓ Greater than or equal to `>=`
- ✓ Less than or equal to `<=`

1.3 Logical Operators

✓ `and` : True if both operands are true

✓ `or` : True if either of the operands is true

✓ `not` : True if operand is false

```
a = False
b = True

print("a and b is ", a and b)
print("a or b is ", a or b)
print("not a is ", not a)
```

```
a and b is  False
a or b is  True
not a is  True
```

1.4 Assignment Opreators

Assignment operators are used to assign values to variables.

✓ `=` gives a variable value

✓ `+=` adds the the original value of the variable by the new value

✓ `-=` subtracts the the original value of the variable by the new value

✓ `*=` multiplies the the original value of the variable by the new value

✓ `/=` divides the the original value of the variable by the new value

```
x = 5
x += 5 # adds 5 th the original x value
x
```

```
10
```

1.5 Identity Operators

They are `is` and `is not`. They are used to check if two values or variables are located on the same path of their memory.

✸ Two values that are equal, does not imply that they are equal.

```
x = 5
y = 10

print(x is not y)
print(x is y)
```

```
True
False
```

1.6 Membership Operators

They are `in` and `not in` and are used to test whether a value or a variable is found in a sequence like a string, tuple, dictionary etc and returns a true or false value.

```
x = "Hi there!"
y = [1, 2, 3, 4]

print("H" in x)
print(1 in y)
print("t" not in x)
print(2 not in y)
```

```
True
True
False
False
```

Chapter 2

Output

```
x = 20
y = 30
print("The value of x is {} and the value of y is {}".format(x, y))
```

```
The value of x is 20 and the value of y is 30
```

```
print("I like {0} and also like {1}".format("Jesus", "Holyghost"))
```

```
I like Jesus and also like Holyghost
```

```
# Using keyword atgs to format
print("Hi {name}, {greet}".format(greet = "welcome", name = "Marcelo"))
```

```
Hi Marcelo, welcome
```

```
x = 5
y = 3.5
z = 2 + 3j

print(x, "is of type", type(x))
print(y, "is of type", type(y))
print(z, "is of type", type(z))
```

```
5 is of type <class 'int'>
3.5 is of type <class 'float'>
(2+3j) is of type <class 'complex'>
```

Chapter 3

Variables

It is possible for two or more variables to represent same value

```
a = 2
b = 2
print(a)
print(b)
```

```
2
2
```

Assigning multiple values in the same line

```
num1, num2, num3, name = 3, 5.7, 10, "Fred"
print(num1)
print(num2)
print(num3)
print(name)
```

```
3
5.7
10
Fred
```

```
a = 1
c = 2
f = a
# Here f refers to the value a is referring to ie 1, so f doesn't refer to the variable a
```

When a is assigned to a different value, f remains to be referring the initial value a was referring to as shown below.

```
a = 3
print(a)
```

```
3
```

```
print(f)
```

```
1
```

3.1 Swapping two variables

```
v1 = "first string"
v2 = "second string"
```

```
temp = v1
v1 = v2
v2 = temp
print(v1, v2)
```

```
second string first string
```

3.2 Inserting variables in strings

```
char_name = "Jonte"
char_age = "34" # rem python can't concatenate strings and int so you must put value your
    value here as a string
print("There was once a man named John")
print("He was 45 years old")
```

```
There was once a man named John
He was 45 years old
```

```
print("There was once a man named " + char_name)
print("He was " + char_age + " years old")
```

```
There was once a man named Jonte
He was 34 years old
```

Chapter 4

Literals

Literals are the raw data given in a variable.

There are three types of literals;

4.1 Numeric literals

- They contain; binary, decimal, octal, Hexadecimal.
- They are unchangeable.
- Int, float and complex literals.
- Sequence of characters
- Single line and multi-line literals

```
num1 = 0b1010 # binary
num2 = 50 # decimal
num3 = 0o310 # octal
num4 = 0x12c # Hexadecimal

num5 = 11.7 # float
num6 = 1.5e2
num7 = 3.2j # complex

print(num1, num2, num3, num4)
print(num5, num6)
```

```
10 50 200 300
11.7 150.0
```

```
msg = "Python is available"
char = "I"
multi_line = """This is
 a multi-line"""
unicode = u"\u00dcnic\u00f6de"
raw_string = r"raw \n string!"

print(msg)
print(char)
```

```
print(multi_line)
print(raw_string)
print(unicode)
```

```
Python is available
I
This is
 a multi-line
raw \n string!
Ünicöde
```

4.2 Boolean Literals

- True==1
- false==0

```
drink = "Available"
food = None

def new_menu(i):
    if i == drink:
        print(drink)
    else:
        print(food)

new_menu(drink)
new_menu(food)
```

```
Available
None
```

4.3 Collections literals

- List
- Tuple
- Dictionary
- Set

```
# Literal collections
fruits = ["fig", "lemon", "banana"]
numbers = (1, 2, 3, 4, 5, 6, 7, 8, 9)
words = {"first word": "Banana", "Second word": "Hi"}
chars = {'A', 'B', 'c'}

print(fruits)
print(words)
print(chars)
print(numbers)
```

```
['fig', 'lemon', 'banana']
{'first word': 'Banana', 'Second word': 'Hi'}
{'A', 'B', 'c'}
(1, 2, 3, 4, 5, 6, 7, 8, 9)
```

Chapter 5

Functions

Functions are:

- Block of code
- Group of related statements
- Pass arguments

```
def function1(x):
  return print(2 * x)
function1(4)
```

```
8
```

5.1 Functions arguments and Parameters.

- Pass data into functions
- Many arguments

```
def f1(name):
    print("Hi " + name)

# Calling f1 function
f1("Jesus")
```

```
Hi Jesus
```

5.1.1 Inserting a collection as an argument

Inserting a list as an argument or a parameter into a function.

```
def fruits(name):
    for fruit in name:
        print(fruit)

fruit_names = ["Apple", "Banana", "Avocado"]
fruits(fruit_names)
```

```
Apple
Banana
Avocado
```

5.2 Return and the expression

Return function:

- ✓ Return a value.
- ✓ To exit a function, and go back to the place from where it was called.

```
def degrees(x):
    return 15 * x

degrees(4)
```

```
60
```

```
# Using multiple arguments
def new_names(name1, name2, name3):
    print("The first name is " + name1)
    print("The second name is " + name2)
    print("The third name is " + name3)

new_names(name1="A", name2="B", name3="C")
```

```
The first name is A
The second name is B
The third name is C
```

- Optional parameters

```
def add_numbers(x, y, z=None):
    if (z == None):
        return x + y
    else:
        return x + y + z

print(add_numbers(1, 2))
print(add_numbers(1, 2, 3))
```

```
3
6
```

- Assigning variable to a function

```
def add_numbers(x, y, z=None, flag=False):
    if (flag):
        print('Flag is true!')
    if (z == None):
        return x + y
    else:
        return x + y + z

print(add_numbers(1, 3, flag=True))
```

```
Flag is true!
4
```

5.3 Arbitrary Arguments

➢ If you do not knw how many keyword arguments that will be passed into the function, use an asterisk before the parameter name name to denote this kind of argument.

```
def trees(*name):
    print("The tree is " + name[0])

trees("Pine", "blue gam")
```

```
The tree is Pine
```

5.4 Recursion

- Solving problems
- Adefined function can call itself

```
def numbers(n):
    if (n > 0):
        result = n + numbers(n - 1)
        print(result)
    else:
        result = 0
    return result

print("\n \n recursion results")
numbers(9)
```

```


 recursion results
1
3
6
10
15
21
28
36
45
```

```
45
```

5.5 `lambda` anynymous function

✓ Anonymous function.

✓ Any number of arguments.

✓ Only have one expression.

Example 1

```
x = lambda i: i + 1
x(3)
```

```
4
```

Example 2

```
multiply = lambda x, y: x * y
multiply(3, 5)
```

```
15
```

Example 3

```
math_fn = lambda x, y, z, w: ((x * y) + z) ** w
math_fn(1, 2, 3, 4)
```

```
625
```

Chapter 6

Strings

A string is a sequence of characters. It is usually written using quotation marks.

```
string1 = 'Welcome'
print(string1)
```

```
Welcome
```

Triple quotes string can extend multiple lines

```
string2 = ''' welcome
to the world of
python programming'''
print(string2)
```

```
 welcome
to the world of
python programming
```

Concatenation of strings

```
print(string1 + string2)
```

```
Welcome welcome
to the world of
python programming
```

Iterating through a string

```
letter_count = 0
for letters in 'Hello world':
  if letters == 'l':
    letter_count += 1
print(letter_count)
```

```
3
```

String membership

```
print('l' in 'hello')
print('l' not in 'hello')
```

```
True
False
```

Built-in functions

```
string = 'university'
# using enumerate()
list(enumerate(string))
```

```
[(0, 'u'),
 (1, 'n'),
 (2, 'i'),
 (3, 'v'),
 (4, 'e'),
 (5, 'r'),
 (6, 's'),
 (7, 'i'),
 (8, 't'),
 (9, 'y')]
```

escape character

```
print("In Jesus' name") # If you want to use single quotes in a string, use double quotes to
    close the string
```

```
In Jesus' name
```

string methods

```
print('gOOD moRning tO all'.lower())
print('gOOD moRning tO all'.upper())
print('gOOD moRning tO all'.title())
print('gOOD moRning tO all'.replace('all', 'everybody'))
```

```
good morning to all
GOOD MORNING TO ALL
Good Morning To All
gOOD moRning tO everybody
```

- Regular expression evaluation: segmenting string looking for patterns

```
firstname = 'Brian'
lastname = 'Chege'

print(firstname + ' ' + lastname)
print(firstname * 3)
print('Brian' in firstname)
```

```
Brian Chege
BrianBrianBrian
True
```

```
firstname = 'Brian Chege Gitu'.split(' ')[0]
lastname = 'Brian Chege Gitu'.split(' ')[-1]
print(firstname)
print(lastname)
```

```
Brian
Gitu
```

- A glimpse of Python string formatting:
  1. Allows you to write a string statement indicating placeholders
  2. Then pass these variables in either named of in order of arguments

```
sales_record = {'price': 3.24,
                'num_items': 4,
                'person': 'Brian'}

sales_statement = '{} bought {} items(s) at a price of {} each for a total of {}'

print(sales_statement.format(sales_record['person'],
                             sales_record['num_items'],
                             sales_record['price'],
                             sales_record['num_items']*sales_record['price']))
```

```
Brian bought 4 items(s) at a price of 3.24 each for a total of 12.96
```

Chapter 7

Set

Set is an unordered collection of unique items in python. It is usually written using curly brackets

- ✓ Unordered.
- ✓ Eliminates duplicates.
- ✓ You can place any type of data you want.

☛ **Example:** `{1, 2, 3, 4, 5}`

Note: Items in a set can not be accessed directly using index since sets objects are not subsctiptable or indexed but a loop can be used.

```
set1 = {"word", "rhema", "peace"} # string
set2 = {1, 2, 3,} # integers
set3 = {False, True, False} # boolean

# set variables or items
print(set1)
print(set2)
print(set3)
# Type of set_1
print(type(set1))
```

```
{'rhema', 'word', 'peace'}
{1, 2, 3}
{False, True}
<class 'set'>
```

```
# Different data types
set4 = {True, "Python", 6}
print(set4)
```

```
{True, 'Python', 6}
```

```
# Create set using set function
set5 = set((True, "yes", 7))
print(set5)
```

```
{'yes', True, 7}
```

```
# Accessing items in a set
for item in set5:
    print(item)
```

```
# Checking item in a set
print(7 in set5)

# add item to a set
set5.add("Good")
print(set5)

# update item in a set
set5.update(["lemon", "cherry"])
print(set5)
```

```
True
{'yes', True, 'Good', 7}
{'yes', True, 7, 'lemon', 'Good', 'cherry'}
```

Other set functions inlcude:

- `set5.remove()` for removing item from a list, gives error if the item is not in the list.
- `set5.discard()` for discarding item from a list, does not give error if the item is not in the list.
- `set5.pop()` for removing the last item from list, but remember set are unordered.
- `set5.clear()` for clearing the items in a list.
- `del set5` for deleting the set completely.

Chapter 8

Lists

A **List** can be written as a list of comma-separated values(items) between square brackets. Items in a list need not have the same type. Creating a list is as simple as putting different comma separated values between square brackets.

A List:

- ✓ Store multiple data types.
- ✓ Ordered and changeable.

☛ **Example:** `[1, 2, 3, 4, 5]`

8.1 Basic list operations

Lists respond to all of the general sequence operations.

8.2 List methods

Table 8.1: List manipulation tools

Method	Description
`append()`	Adds an element at the end of the list
`clear()`	Removes all the elements from the list
`copy()`	Returns a copy of the list
`count()`	Returns the number of elements with the specified value
`extend()`	Add the elements of a list (or any iterable), to the end of the current list
`index()`	Returns the index of the first element with the specified value
`insert()`	Adds an element at the specified position
`pop()`	Removes the element at the specified position
`remove()`	Removes the item with the specified value
`reverse()`	Reverses the order of the list
`sort()`	Sorts the list

The above list methods are used by typing listname.method eg `list1.append()`

8.3 Built-in list functions

`len(list)` Gives the total length of the list

`max(list)` Returns item from the list with max value

`min(list)` Returns item from the list with min value

`list(seq)` Converts a sequence into list

```
list1 = ['physics', 'chemistry', 1997, 2000]
list2 = [1, 2, 3, 4, 5]
list3 = ['a', 'b', 'c']
list4 = [2, 'abs', 3.5, True, "Hi"]
print(list1)
print(list2)
print(list3)
print(list4)
print(type(list4))
```

```
['physics', 'chemistry', 1997, 2000]
[1, 2, 3, 4, 5]
['a', 'b', 'c']
[2, 'abs', 3.5, True, 'Hi']
<class 'list'>
```

- Extract or access a list

```
print(list4[0]) #notice it starts from zero
print(list4[1])
```

```
2
abs
```

```
# still on extracting
print(list4[-1]) # but if you add -ve, it starts with the last element
print(list4[-2])
print(list4[0:2]) #  NOTICE!  It does not include index 2!!!!!!
```

```
Hi
True
[2, 'abs']
```

- Update a list

```
list3[2] = 'd'
print(list3)
```

```
['a', 'b', 'd']
```

- Delete elements in a list

```
del list1[1]
print(list1)
```

```
['physics', 1997, 2000]
```

- To check if a specific item is in the list

```
list4 = [2, 'abs', 3.5, True, "Hi"]
if "Hi" in list4:
    print("correct")
```

```
correct
```

- Another simpler way of checking items in a list that returns a boolean value

```
'abs'  in list4
```

```
True
```

```
if "There" in list4:
    print("correct")
else:
    print("wrong")
```

```
wrong
```

- Inserting list in another list

```
list4[0] = ["Apple", "Watermelon"]
print(list4)
```

```
[['Apple', 'Watermelon'], 'abs', 3.5, True, 'Hi']
```

- Inserting using insert function

```
list4.insert(1, ["Apple", "Watermelon"]) # as you run more times, it keeps on inserting
print(list4)
```

```
[['Apple', 'Watermelon'], ['Apple', 'Watermelon'], 'abs', 3.5, True, 'Hi']
```

- append function

```
x = [1, 2, 3, 4]
x.append(5)
print(x)
```

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

- The + operator adds lists

```
[1, 2] + [3, 4]
```

```
[1, 2, 3, 4]
```

- The * character repeats values of a list

```
[1] * 3
```

```
[1, 1, 1]
```

- Using for loop inside a list

```
my_list = []
for number in range(0, 10):
    if number % 2 == 0:
        my_list.append(number)

my_list
```

```
[0, 2, 4, 6, 8]
```

```
my_list = [number for number in range(0, 10) if number % 2 == 0]
my_list
```

```
[0, 2, 4, 6, 8]
```

Chapter 9

Decision making statements

Anticipate possible conditions occurring while execution of the program and specify instructions to be executed for each condition.

Decision statement evaluates logical expression which produces TRUE or FALSE as outcome.

Specify statements to execute if outcome is TRUE or FALSE.

non-zero and non-null values as TRUE zero or null is assumed as FALSE.

Table 9.1: Decision making statements

Statement	**Description**
if statement	Consists of a boolean expression followed by one or more statements
if else statement	an if statement followed by an optional else statement, which executes when the boolean expression is FALSE
nested if statement	One if or else if statement inside another if or else if statement(s)

9.1 If statement

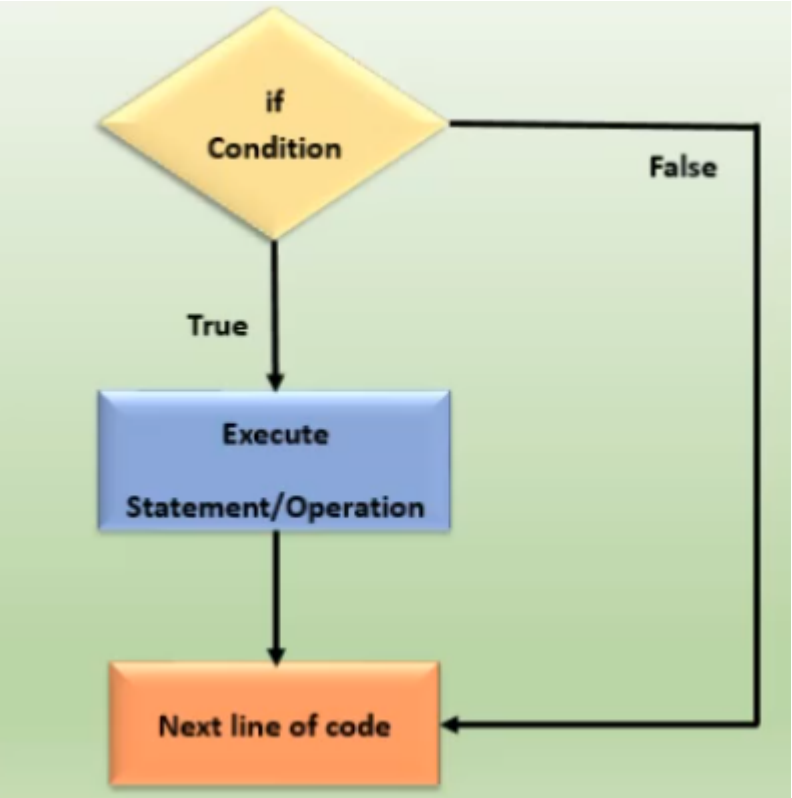


Figure 9.1: If flow chart

```
a = 1
b = 2
if a < b:
  print('a is less than b')
```

```
a is less than b
```

9.2 If else statements

An optional else combines with if statement

if true, then immediately following block is executed

if not true(false), block following else is executed

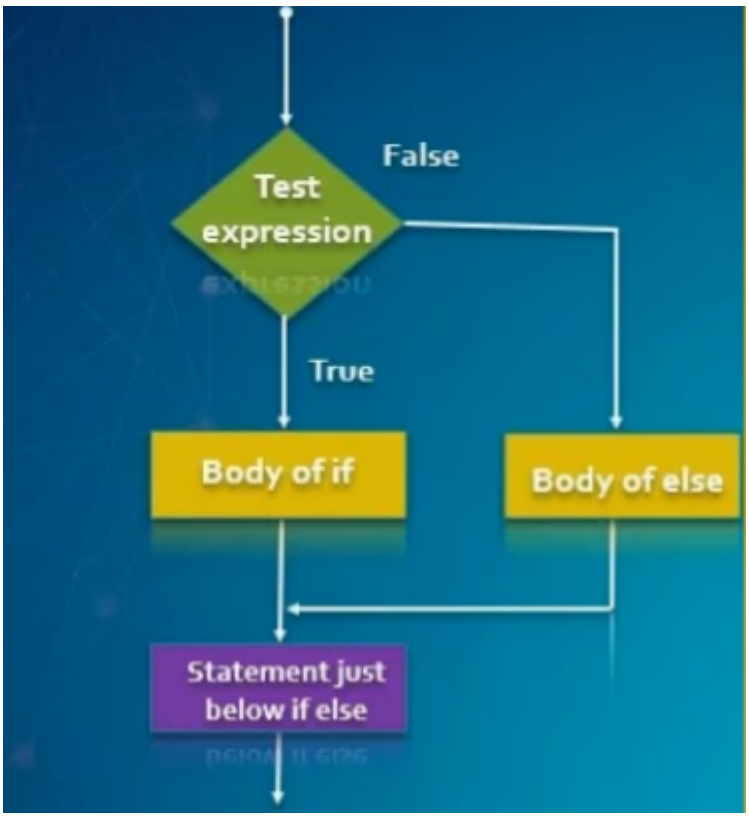


Figure 9.2: If else flow chart

```
c = 4
d = 4
if c < d:
  print("c is less than d")
else:
  print("c is not less than d")
```

```
c is not less than d
```

9.3 If elif else statement

`elif` statement allows you to check multiple expressions.

There can be any number of `elif` statements following an if

`elif` statement is also optional

```
e = 16
f = 8
if e < f:
  print('e is less than f')
elif e == f:
  print('e is equal to f')
elif e > f + 10:
  print('e is greater than f by more than ten')
else:
  print('e is greater than f')
```

```
e is greater than f
```

9.4 Nested if statement

Conditional statements can be nested inside one another

```
x = -2
if x >= 0:
    if x == 0:
        print("x is zero")
    else:
        print("x is positive")
else:
    print("x is negative")
```

```
x is negative
```

9.5 Single statements suite

If the suite of an if clause consists only of a single line, it may go on the same line as the header statement.

```
marks = 51
if marks >= 50: print('pass')
else: print('fail')
```

```
pass
```

Chapter 10

Loops

A loop or iteration is one or more sequential statements executed repeatedly until a certain condition met.

10.1 While loop

A while loop statement executes a target statement as long as a given condition is true.

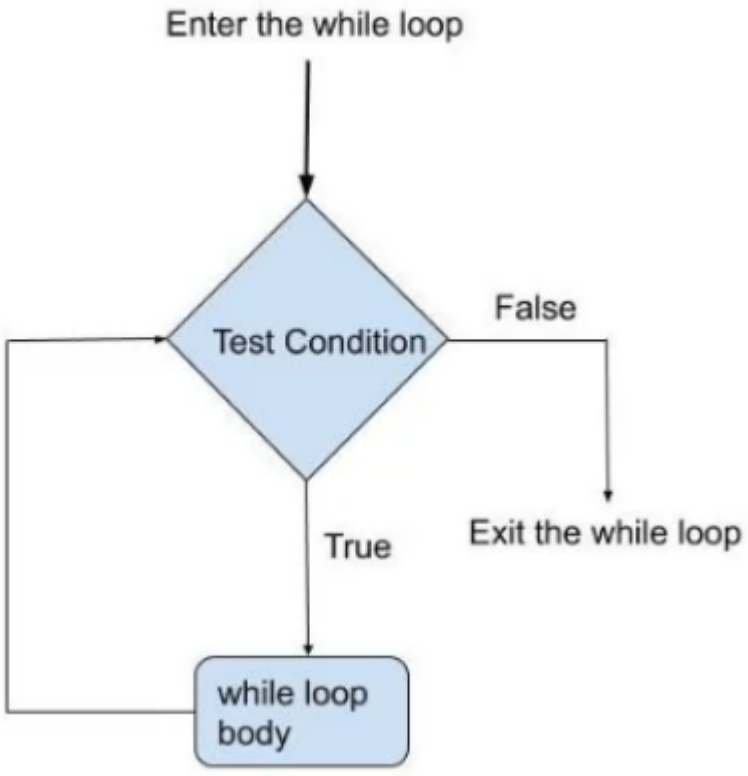


Figure 10.1: While loop

```
count = 0
while count < 10:
```

```
  count = count + 1
  print('count', count)
```

```
count 1
count 2
count 3
count 4
count 5
count 6
count 7
count 8
count 9
count 10
```

10.1.1 While Loop else statement

When the else statement is used with a while loop, the else statement is executed when the condition becomes false.

```
count = 0
while count < 10:
  print(count, 'is less than 10')
  count = count + 1
else:
    print(count, 'is not less than 10')
```

```
0 is less than 10
1 is less than 10
2 is less than 10
3 is less than 10
4 is less than 10
5 is less than 10
6 is less than 10
7 is less than 10
8 is less than 10
9 is less than 10
10 is not less than 10
```

10.2 For loop

The for loop has the ability to iterate over the items of any sequence,such as a list or a string. If a sequeence contains an expression list, it is evaluated first then, first item in sequence is assigned to iterating variable.

Next, the statements block is executed Each item in the list is assigned to iterating variable statement(s) block is executed until the entire sequence is exhausted.

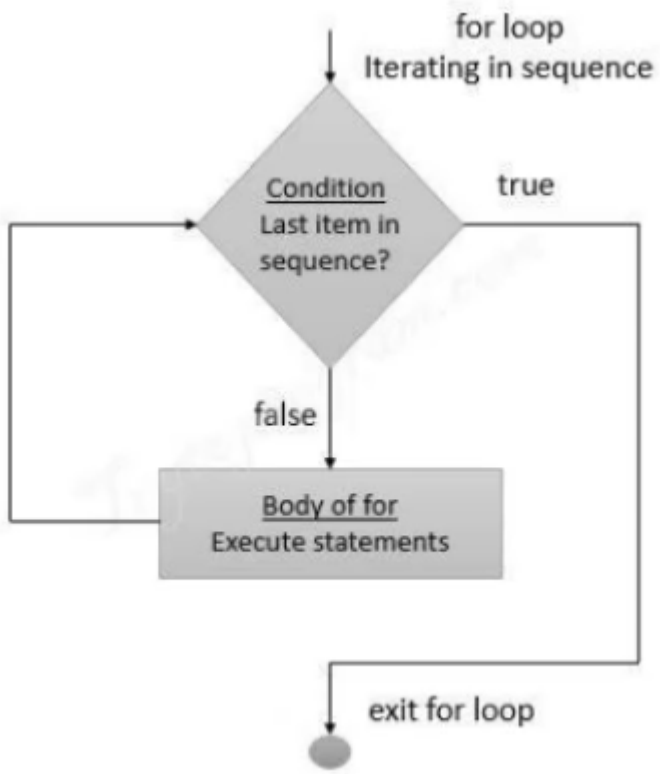


Figure 10.2: For loop

```
data = [1, 2, 3, 4, 5]
for i in data:
  print(i)
```

```
1
2
3
4
5
```

10.2.1 For loop using `range()` function

`range()` function generates list of numbers. It takes 0 as starting parameter but you can specify the starting parameter.

For example: `range(2, 7)` returns a sequence of numbers starting from 2 and going to 7, for loop iterates through these numbers.

```
for i in range(1, 6, 2):
  print(i)
```

```
1
3
5
```

10.2.2 For loop else statement

If the else statement is used with a for loop, the else statement is executed when the loop has exhausted iterating the list.

```
count = 0
for count in range(10):
  print(count)
else:
  print('end of loop')
```

```
0
1
2
3
4
5
6
7
8
9
end of loop
```

10.3 Nested loops

A nested loop is any type of loop inside of any other type of loop.

```
for i in range(1, 11):
  for j in range(1, 11):
    k = i * j
    print(k, end = ' ')
  print()
```

```
1 2 3 4 5 6 7 8 9 10
2 4 6 8 10 12 14 16 18 20
3 6 9 12 15 18 21 24 27 30
4 8 12 16 20 24 28 32 36 40
5 10 15 20 25 30 35 40 45 50
6 12 18 24 30 36 42 48 54 60
7 14 21 28 35 42 49 56 63 70
8 16 24 32 40 48 56 64 72 80
9 18 27 36 45 54 63 72 81 90
10 20 30 40 50 60 70 80 90 100
```

10.4 Loop control statements

Loop control statements change execution from normal sequence. When execution leaves a scope, all automatic objects in that cope are destroyed.

Python supports the following loop control statements:

Table 10.1: Loop control statements

Control statment	Description
`break`	Terminate the loop statement and transfers execution to the statement immediately following the loop
`continue`	Causes the loop to skip the remainder of its body and immediately retest its condition prior to re-iterating
`pass`	The pass statement is used when a statement is required syntactically but you don't want any command or code to execute

10.4.1 Break statement

The break statement terminates the current loop and resumes execution at the next statement.

In case of nested loops, the break statement will stop the execution of the innermost loop and start executing the next line of code after the block.

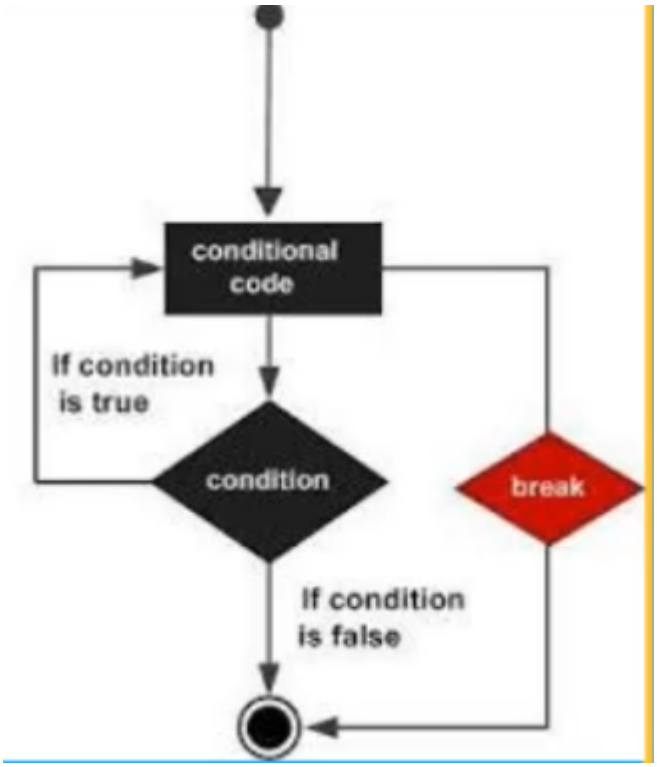


Figure 10.3: Break statement flow chart

```
var = 10
while var > 0:
  var = var - 1
  if var == 5:
    continue
  print(var)
```

```
9
8
7
6
4
3
2
1
0
```

10.4.2 Pass statement

The pass statement is used when a statement is required syntactically but you do not want any command or code to execute The pass statement is a null operation,nothing happens when it executes.

Chapter 11

Tuple

A tuple is a sequence of immutable variables python objects. It is usually written using parentheses.

Characteristics of lists:

- ✓ Ordered sequence.
- ✓ Immutable sequence.
- ✓ Protect data.
- ✓ Faster than lists.
- ✓ One tuple can contain many data types together. Eg: `("Hi", True, 1, 3.8)`

```
tup1 = ('physics', 'chemistry', 1997, 2000,)
tup2 = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11)
print(tup1, tup2)
print(type(tup1))
print(tup1[0:2])
```

```
('physics', 'chemistry', 1997, 2000) (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11)
<class 'tuple'>
('physics', 'chemistry')
```

```
# Creating tuple using tuple()
new_t = tuple(("Apple", "Google", "Microsoft"))
print(new_t)
```

```
('Apple', 'Google', 'Microsoft')
```

11.1 Built-in tuple function

`len(tuple)` : Gives the total length of the tuple

`max(tuple)` : Returns item from the tuple with max value

`min(tuple)` : Returns item from the tuple with min value

`tuple` : Converts a sequence into a tuple

Chapter 12

Dictionary

A dictionary is a Comma separated pairs of key and value enclosed in curly brackets. Each key is separated from its value by a colon (:). Keys are unique within a dictionary while values may not be.

Properties of a dictionary:

- ✓ The values of a dictionary can be of any type.
- ✓ Unordered and changeable.
- ✓ No duplicates.
- ✓ Keys must be of an immutable data type such as strings,numbers, or tuples.

➢ **Syntax:** `{key1: value1, key2: value2}`

```
newcars = {
    "Brand": "BMW",
    "Model": "XS",
    "Year": 2017
}
print(newcars)
type(newcars)
```

```
{'Brand': 'BMW', 'Model': 'XS', 'Year': 2017}
```

```
dict
```

```
print(newcars["Brand"])
x = newcars.get("Brand")
print(x)

x1 = newcars.keys()
print(x1) # printing the keys in the dictionary
```

```
BMW
BMW
dict_keys(['Brand', 'Model', 'Year'])
```

```
# Add a new item to the dictionary
newcars["Color"] = "black"
print(newcars)

# printing only values of the dictionary
x = newcars.values()
print(x)
```

```
{'Brand': 'BMW', 'Model': 'XS', 'Year': 2017, 'Color': 'black'}
dict_values(['BMW', 'XS', 2017, 'black'])
```

```
# Printing all items in the dictionary
newcars.items()
```

```
dict_items([('Brand', 'BMW'), ('Model', 'XS'), ('Year', 2017), ('Color', 'black')])
```

```
# accessing items in the dictionary
if "Color" in newcars:
    print("Yes")
```

```
Yes
```

```
dict1 = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
print(dict1)
print(type(dict1))
print(dict1['Name'])
```

```
{'Name': 'Zara', 'Age': 7, 'Class': 'First'}
<class 'dict'>
Zara
```

```
# Update
dict1['Age'] = 8 # Update existing entry
dict1['School'] = "SIBEA" # Add new entry
print(dict1)
```

```
{'Name': 'Zara', 'Age': 8, 'Class': 'First', 'School': 'SIBEA'}
```

```
# Delete
del dict1['Name'] # remove entry with key 'Name'
print ("directory after removing element" , dict1)
dict1.clear() # remove all entries in dict
print ("directory after clear" , dict)
del dict1 # delete entire dictionary
```

```
directory after removing element {'Age': 8, 'Class': 'First', 'School': 'SIBEA'}
directory after clear <class 'dict'>
```

Properties of Dictionaries are:

- Dictionary values have no restrictions
- They can be any arbitary python object, either standard objects or user - defined objects
- However, same is not true for the keys
- More than one entry per key not allowed
- When duplicate keys encountered during assignment, the last asignment wins
- Keys must be immutable

- Strings, numbers or tuples as dictionary keys but something like [‘key’] is not allowed

```
dict1 = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
print(len(dict1)) # Gives the length of the dictionary
str(dict1) # Produces the string representation of the dictionary
```

```
3
```

```
"{'Name': 'Zara', 'Age': 7, 'Class': 'First'}"
```

Dictionary Methods

dict.clear() Removes all elements of dictionary *dict*
dict.copy() Returns a shallow copy of dictionary *dict*
dict.fromkeys() Create a new dictionary with keys from seq and values *set* to *value*.
dict.get(key, default=None) For *key* key, returns value or default if key not in dictionary
dict.has_key(key) Returns *true* if key in dictionary *dict*, *false* otherwise
dict.items() Returns a list of *dict*'s (key, value) tuple pairs
dict.keys() Returns list of dictionary dict's keys
dict.setdefault(key, default=None) Similar to get(), but will set dict[key]=default if *key* is not already in dict
dict.update(dict2) Adds dictionary *dict2*'s key-values pairs to *dict*
dict.values() Returns list of dictionary *dict*'s values

Figure 12.1: Dictionary methods

- Iterating all the keys in a dictionary

```
dict1 = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}

for key in dict1.keys():
    print(key)
```

```
Name
Age
Class
```

- Iterating all the values and ignore the keys

```
for value in dict1.values():
    print(value)
```

```
Zara
7
First
```

- Iterate both keys and values in a dictionary using items function

```
for key, value in dict1.items():
    print(key, value)
```

```
Name Zara
Age 7
Class First
```

Chapter 13

Classes and Objects

- ✓ Python is an object oriented language.
- ✓ Everything in python is an object.
- ✓ Class is an object contructor.
- ✓ All classes have a function called `__init__`. This can be used to assign values to object properties or other operations.

13.1 Classes

```
class NewClass:
    num = 10

print(NewClass)
```

```
<class '__main__.NewClass'>
```

```
p1 = NewClass() # created a new object base on the class created
print(p1.num)
```

```
10
```

```
# Using init
class Person:
    def __init__(self, name, age): # self parameter is a reference to the current instance of
     the class and is used to access variables that are in this class
        self.name = name # in order to access the variable name, self is used
        self.age = age

obj1 = Person("Don", 56)

print(obj1.name)
print(obj1.age)
```

```
Don
56
```

13.2 Object Methods

- Ojbects can also contain methods and objects are functions that belong to the object.

```
# Creating a simple method in the Pesron class
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def hi(self):
        print("Hi, my name is " + self.name)

obj1 = Person("Don", 56)
obj1.hi()
```

```
Hi, my name is Don
```

- Properties on objects can also be defined

```
# modify obj1
obj1.age = 50
print(obj1.age)
```

```
50
```

13.3 The `map()` function

```
store1 = [10.00, 11.00, 12.34, 2.34]
strore2 = [9.00, 11.10, 12.34, 2.01]
cheapest = map(min, store1, strore2)
cheapest
```

```
<map at 0x22d872a5b20>
```

Gives a map object, in which you can use for loop to get your answer. This allowa us to have very efficient memory management.

```
for item in cheapest:
    print(item)
```

```
9.0
11.0
12.34
2.01
```

Chapter 14

Files Handling

- `Open("filename", "mode")`
- **Modes:**
  1. r This refers to read, and it is the default.
  2. a This refers for appending, open files for appending, also create a file if it doesn't exist
  3. w This refers to write, open files for writing, also create a file if it doesn't exist
  4. x This refers to create, it creates a specified file. It has two additional options, t for creating a text and b for binary.

```
f = open("demo.txt", "r")
print(f.read())
print(f.close())
```

```
Hello, im a demo text

line two

line threeThis is an update to the existing text file
None
```

```
f = open("demo.txt", "r")
print(f.readline()) # prints blank after reading the whole file ie f.readline()
```

```
Hello, im a demo text
```

Writing to an existing file

```
f = open("demo.txt", "a")
f.write("This is an update to the existing text file")
f.close() # Closing the file so as to open it as read-only
```

```
f = open("demo.txt", "r")
print(f.read())
```

```
Hello, im a demo text

line two

line threeThis is an update to the existing text fileThis is an update to the existing text
    file
```

Chapter 15

Importing Modules

- Importing modules, libraries.
- A python module is a python file that ends with .py
- Built-in modules
- A module can contain a function
- A module can be of any type, array,dictionary, tuple….etc.
- Can be renamed using as, eg: `import module as md`

```
# created a module1.py file, it will be imported as a module
import module1
module1.greeting("Rona")
```

```
Hello Rona
```

```
# importing a dictionary
import module2 as m2
x = m2.person["email"]
print(x)
```

```
test@example.com
```

```
# importing a built-in module
import math
a = dir(math) # dir() returns names of functions in the specified module
print(a)
```

```
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin', '
    asinh', 'atan', 'atan2', 'atanh', 'ceil', 'comb', 'copysign', 'cos', 'cosh', 'degrees', '
    dist', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp',
     'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'isqrt'
    , 'lcm', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'nextafter',
    'perm', 'pi', 'pow', 'prod', 'radians', 'remainder', 'sin', 'sinh', 'sqrt', 'tan', 'tanh'
    , 'tau', 'trunc', 'ulp']
```

importing only parts of the module

```
from module2 import person
print(person["name"])
```

```
Brian
```

15.1 Importing from SL

```
import math

print(math.pi)
print(math.e)
```

```
3.141592653589793
2.718281828459045
```

If you want to import functions only:

```
from math import pi, e

print(pi)
print(e)
```

```
3.141592653589793
2.718281828459045
```

```
import sys

print(sys.path)
```

```
["d:\\Chege Learners' Hub\\Python\\Labs\\Essentials For Python Programming", 'c:\\Users\\
    ryanc\\anaconda3\\python39.zip', 'c:\\Users\\ryanc\\anaconda3\\DLLs', 'c:\\Users\\ryanc\\
    anaconda3\\lib', 'c:\\Users\\ryanc\\anaconda3', '', 'c:\\Users\\ryanc\\anaconda3\\lib\\
    site-packages', 'c:\\Users\\ryanc\\anaconda3\\lib\\site-packages\\win32', 'c:\\Users\\
    ryanc\\anaconda3\\lib\\site-packages\\win32\\lib', 'c:\\Users\\ryanc\\anaconda3\\lib\\
    site-packages\\Pythonwin', 'c:\\Users\\ryanc\\anaconda3\\lib\\site-packages\\IPython\\
    extensions', 'C:\\Users\\ryanc\\.ipython']
```

Chapter 16

NumPy

- ✓ NumPy is a numerical library for Python that allows for extremely fast data generation and handling.
- ✓ It utilizes arrays which can efficiently store data (much better than a built-in normal python list).
- ✓ It has functions for linear algebra, Fourier transform and matrices.

```
import numpy as np
print(np.__version__)
```

```
1.23.5
```

16.1 NumPy arrays

```
arr1 = np.array([1, 2, 3, 4, 5])
print(arr1)
print(type(arr1))
```

```
[1 2 3 4 5]
<class 'numpy.ndarray'>
```

- 0-Dimensional array

```
arr3 = np.array(30)
print(arr3)
```

```
30
```

- 1- Dimensional array

```
arr4 = np.array([1, 2, 3, 4])
print(arr4)

np.zeros(4) # the point here is for accuracy.
np.ones(3)
```

```
[1 2 3 4]
```

```
array([1., 1., 1.])
```

- 2-Dimensional array (matrix)

```
arr5 = np.array([[1, 2, 3], [4, 5, 6]])
print(arr5)

np.ones((3, 3))
```

```
[[1 2 3]
 [4 5 6]]
```

```
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

- Iterating over two dimensional arrays

```
for i in arr5:
    print(i)
```

```
[1 2 3]
[4 5 6]
```

```
for i in arr5:
    for j in i:
        print(j)
```

```
1
2
3
4
5
6
```

- **Array Shape**
  - NumPy arrays have a shape attribute which returns a tuple:
    - First value of the tuple gives the number of dimensions in the array
    - Second value of the tuple gives the number of elements in each dimension

```
arr5 = np.array([[1, 2, 3], [4, 5, 6]])
print(arr5.shape)
```

```
(2, 3)
```

- 3-Dimensional array

```
arr6 = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
print(arr6)
```

```
[[[ 1  2  3]
  [ 4  5  6]]

 [[ 7  8  9]
  [10 11 12]]]
```

- Iterating over 3-D arrays

```
for i in arr6: # read as: for matrix in arr6
    for j in i: # read as: for row in matrix i
        for k in j: # read as: for scalar in row j
            print(k)
```

```
1
2
3
4
5
6
7
8
9
10
11
12
```

```
np.linspace(0, 5, 15) # its different from arange, in that it has start and stop values and
    the next value is the number of equally spade values
```

```
array([0.        , 0.35714286, 0.71428571, 1.07142857, 1.42857143,
       1.78571429, 2.14285714, 2.5       , 2.85714286, 3.21428571,
       3.57142857, 3.92857143, 4.28571429, 4.64285714, 5.        ])
```

- Identity matrix : is a square matrix

```
np.eye(4)
```

```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

- random number generator of a uniform distribution

```
np.random.rand(5, 4) # a 5 x 4 matrix with random numbers from a uniform distribution
```

```
array([[0.35921494, 0.79724859, 0.90576334, 0.81207866],
       [0.06733697, 0.06576014, 0.56079566, 0.44254108],
       [0.58898953, 0.83239765, 0.22605324, 0.94865553],
       [0.75894874, 0.41362042, 0.86296112, 0.53155822],
       [0.29259861, 0.83679089, 0.09701284, 0.47360349]])
```

- random number generator of a standard normal distribution

```
np.random.randn(5, 4)
```

```
array([[-0.82417037,  0.74732258, -1.4055911 ,  0.31655887],
       [ 0.62974636, -0.99764014, -1.4158089 , -1.49006231],
       [ 0.20067351, -1.28912409, -1.04928644, -0.34473435],
       [-1.31833813,  0.15816877,  0.33848458,  0.96306509],
       [-0.77662948,  0.23413739, -1.13847024, -0.83622048]])
```

- random Interger

```
np.random.randint(1, 100, 9) # returns a random interger from low to high, high is exclusive.
     the 9 is the number of intergers
```

```
array([89, 26, 44, 36, 71, 72, 78, 89, 80])
```

- Checking number of dimensions

```
print(arr3.ndim)
print(arr4.ndim)
print(arr5.ndim)
print(arr6.ndim)
```

```
0
1
2
3
```

- Higher dimensional array

```
arr1 = np.array([1, 2, 3, 4], ndmin = 5)
print(arr1)
print(arr1.ndim)
```

```
[[[[[1 2 3 4]]]]]
5
```

- Accessing elements from 1-D array

```
arr1 = np.array([10, 20, 30, 40])
print(arr1[0])
print(arr1[2])
```

```
10
30
```

- Accessing elements from 2-D array

```
arr5 = np.array([[1, 2, 3], [4, 5, 6]])
print("The second element in 1st dim is:", arr5[0, 1])
print("The third element in 2nd dim is:", arr5[1, 2])
```

```
The second element in 1st dim is: 2
The third element in 2nd dim is: 6
```

- Accessing elements from 3-D array

```
arr6 = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
print("The third element in the second array of the first array is:", arr6[0, 1, 2])
```

```
The third element in the second array of the first array is: 6
```

16.1.1 Slicing indexes in arrays

Slicing 1-Dimensional arrays

```
new_arr = np.array([1, 2, 3, 4, 5, 6])
print(new_arr[1:4]) # does not include index 4
```

```
[2 3 4]
```

```
print(new_arr[3:])
print(new_arr[:3])
print(new_arr[-4: -1])
print(new_arr[1:4:2]) # 1 to 4 and step by 2
```

```
[4 5 6]
[1 2 3]
[3 4 5]
[2 4]
```

Slicing 2-Dimensional arrays

```
new_arr2 = np.array([[1, 2, 3, 4, 5], [10, 20, 30, 40, 50]])
print(new_arr2[1, 1:4])
```

```
[20 30 40]
```

```
print(new_arr2[0:2, 2])
```

```
[ 3 30]
```

```
print(new_arr2[0:2, 1:4])
```

```
[[ 2  3  4]
 [20 30 40]]
```

Numpy has various data types:

- ✓ `i` - integer
- ✓ `b` - boolean
- ✓ `u` - unsigned integer
- ✓ `f` - float
- ✓ `c` - complex number
- ✓ `m` - timedata
- ✓ `M` - datetime
- ✓ `O` - object
- ✓ `S` - string
- ✓ `U` - unicode string
- ✓ `V` - fixed chunk of memory for other type

These options are input in the `dtype` parameter.

```
new_arr3 = np.array([1, 2, 3])
new_arr4 = np.array(["banana", "fig", "orange"], dtype = "S")
print(new_arr3.dtype)
print(new_arr4.dtype)
```

```
int32
|S6
```

16.2 Numpy Mathematical Operations

```
a = np.array([1, 2, 3, 4, 5])
b = a + 2
print(b)
```

```
[3 4 5 6 7]
```

```
b = np.array([10, 20, 30, 40, 50])
c = a + b
print(c)
```

```
[11 22 33 44 55]
```

- Multiplying an array by a scalar integer

```
d = 3 * c
print(d)
```

```
[ 33  66  99 132 165]
```

- Elementwise product of arrays: use `*`

```
A = np.array([[1, 1], [0, 1]])
B = np.array([[2, 0], [3, 4]])
A * B
```

```
array([[2, 0],
       [0, 4]])
```

- Matrix product of arrays: use `@` sign or numpy's `dot()` function

```
A @ B
```

```
array([[5, 4],
       [3, 4]])
```

```
np.dot(A, B)
```

```
array([[5, 4],
       [3, 4]])
```

- Exponential function on an array

```
np.exp(a)
```

```
array([  2.71828183,   7.3890561 ,  20.08553692,  54.59815003,
       148.4131591 ])
```

- Aggregation functions of numpy 1-D arrays

```
a = np.array([1, 2, 3, 4, 5])
print('Sum of 1-D array a:', a.sum())
print('Max of 1-D array a:', a.max())
print('Min of 1-D array a:', a.min())
print('Mean of 1-D array a:', a.mean())
```

```
Sum of 1-D array a: 15
Max of 1-D array a: 5
Min of 1-D array a: 1
Mean of 1-D array a: 3.0
```

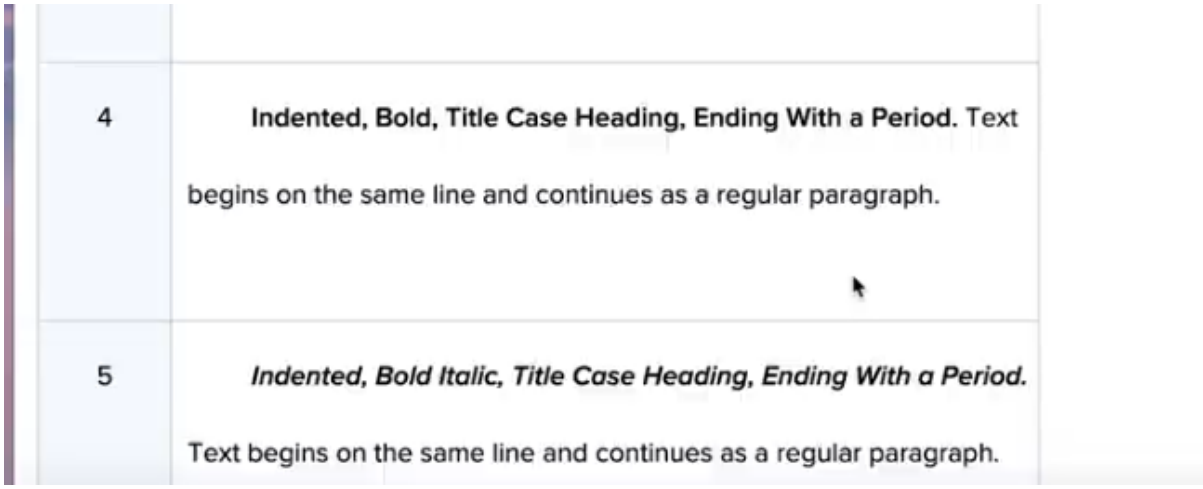
- Aggregation functions of numpy 2-D arrays:
  - We can do the same thing for each row or column with a dim of 3x5

```
c = np.arange(1, 16, 1).reshape(3, 5)
c
```

```
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10],
       [11, 12, 13, 14, 15]])
```

- You can think of these arrays as just a giant ordered list of numbers, and the *shape* of the array, the number of rows and columns, is just an abstraction used for a particular purpose.
- Let's look at the its application in creating images

```
from PIL import Image
from IPython.display import display

im = Image.open("C:/Users/ryanc/Pictures/APA Heading levels 4 and 5.png")
display(im)
```



- We can convert this image into a numpy array

```
array = np.array(im)
print(array.shape)
array
```

```
(258, 645, 4)
```

```
array([[[129, 117, 163, 255],
        [129, 115, 162, 255],
        [127, 114, 158, 255],
        ...,
        [255, 255, 255, 255],
        [255, 255, 255, 255],
```

```
       [255, 255, 255, 255]],

      [[130, 118, 164, 255],
       [130, 116, 162, 255],
       [128, 114, 158, 255],
       ...,
       [255, 255, 255, 255],
       [255, 255, 255, 255],
       [255, 255, 255, 255]],

      [[143, 128, 172, 255],
       [142, 126, 169, 255],
       [140, 125, 166, 255],
       ...,
       [255, 255, 255, 255],
       [255, 255, 255, 255],
       [255, 255, 255, 255]],

      ...,

      [[149, 129, 152, 255],
       [149, 129, 151, 255],
       [149, 130, 151, 255],
       ...,
       [245, 245, 245, 255],
       [245, 245, 245, 255],
       [245, 245, 245, 255]],

      [[148, 129, 151, 255],
       [148, 129, 150, 255],
       [148, 130, 150, 255],
       ...,
       [244, 244, 244, 255],
       [244, 244, 244, 255],
       [244, 244, 244, 255]],

      [[147, 130, 149, 255],
       [147, 130, 149, 255],
       [147, 131, 149, 255],
       ...,
       [244, 244, 244, 255],
       [244, 244, 244, 255],
       [244, 244, 244, 255]]], dtype=uint8)
```

Chapter 17

Pandas Library For Data Science

Pandas is a Python library for data manipulation and analysis.
It allows Exploring, Cleaning and Processing tabular data.

It provides two ways for storing data:

- ✓ Series, which is one dimensional data structure.
- ✓ Data Frame, which is two dimensional data structure.

Creating a data frame using lists:

```
import pandas as pd
```

```
L1 = [['Apple', 'Red'], ['Banana', 'Yellow'], ['Black','Orange']]
df = pd.DataFrame(L1, columns=['Fruit', 'Color'])
df
```

	Fruit	Color
0	Apple	Red
1	Banana	Yellow
2	Black	Orange

Creating a data frame using Dictionary

```
Dictionary = {'Fruit':['Apple', 'Banana', 'Orange'], 'Color':['Red', 'Yellow', 'Orange']}
df1 = pd.DataFrame(Dictionary)
df1
```

	Fruit	Color
0	Apple	Red
1	Banana	Yellow
2	Orange	Orange

17.1 Loading csv file as a data frame

cel data

17.1.1 Data and Data Description

Table 17.3: cel data variable names and there description

Variable name	Description
thomas_name	Name of the member
congress	number of the congress (there is a new congress every two years)
year	year of the start of the congress
st_name	State abbreviation for the member's district
cd	congressional district number
dem	0/1 indicator for whether the member is a democrat
elected	year the member was elected
female	0/1 indicator for whether the member is female
votepct	the percent of the vote the MC won in the election for this congress
dwnom1	DW-Nominate score indicative member ideology. Higher is more conservative
deleg_size	How many MCs are in the member's state delegation?
speaker	Is the member the Speaker of the House? 0/1
subchr	Is the member the chair of a congressional subcommittee?
afam	Is the member African American? 0/1
latino	Is the member latino?
power	Is the member on a "powerful" committee in Congress?
chair	Is the member a chair of a full committee?
state_leg	Was the member a state legislator prior to being elected to congress?
state_leg_prof	How professionalized is the state legislature in the member's state? Higher is more professional
majority	Is the member in the majority in this congress? 0/1
maj_leader	Is the member a majority leader in this congress? 0/1
min_leader	Is the member a minority leader in this congress? 0/1
meddist	How far away is the member from the chamber median dwnom1 score?
meddist	How far away is the member from the majority median dwnom1 score?
all_bills	How many bills did the member introduce in this congress?
all_aic	How many bills did the member introduce that get action in a committee in this congress?
all_abc	How many bills did the member introduce that get action beyond the committee state in this congress?
all_pass	How many bills did the member introduce that passed out of the House in this congress?
all_law	How many bills did the member introduced that became law in this congress?
les	Volden and Wiseman's legislative effective score (LES). Higher means the member is more effective.
seniority	How many term has the member been in congress, including the current term

```
df = pd.read_csv('cel_data.csv')
```

```
df1 = pd.read_csv('cel_data.csv')
df.head()
```

	Unnamed: 0	thomas_num	thomas_name	icpsr	congress	year	st_name	cd	dem	
0	1	1.0	Abdnor, James	14000.0	93	1973	SD	2.0	0	
1	2	2.0	Abzug, Bella	13001.0	93	1973	NY	20.0	1	
2	3	3.0	Adams, Brock	10700.0	93	1973	WA	7.0	1	
3	4	4.0	Addabbo, Joseph	10500.0	93	1973	NY	7.0	1	
4	5	6.0	Alexander, Bill	12000.0	93	1973	AR	1.0	1	

Changing the Index column

It is possible to make one the column of the data to be the index column

```
df.set_index('thomas_name').head()
```

thomas_name	Unnamed: 0	thomas_num	icpsr	congress	year	st_name	cd	dem	elec
Abdnor, James	1	1.0	14000.0	93	1973	SD	2.0	0	197
Abzug, Bella	2	2.0	13001.0	93	1973	NY	20.0	1	197
Adams, Brock	3	3.0	10700.0	93	1973	WA	7.0	1	196
Addabbo, Joseph	4	4.0	10500.0	93	1973	NY	7.0	1	196
Alexander, Bill	5	6.0	12000.0	93	1973	AR	1.0	1	196

```
df.head()
```

	Unnamed: 0	thomas_num	thomas_name	icpsr	congress	year	st_name	cd	dem	
0	1	1.0	Abdnor, James	14000.0	93	1973	SD	2.0	0	
1	2	2.0	Abzug, Bella	13001.0	93	1973	NY	20.0	1	
2	3	3.0	Adams, Brock	10700.0	93	1973	WA	7.0	1	
3	4	4.0	Addabbo, Joseph	10500.0	93	1973	NY	7.0	1	
4	5	6.0	Alexander, Bill	12000.0	93	1973	AR	1.0	1	

Using `inplace` function to change the original data structure

```
df1.set_index('thomas_name', inplace=True)
df1.head()
```

thomas_name	Unnamed: 0	thomas_num	icpsr	congress	year	st_name	cd	dem	elec
Abdnor, James	1	1.0	14000.0	93	1973	SD	2.0	0	197
Abzug, Bella	2	2.0	13001.0	93	1973	NY	20.0	1	197
Adams, Brock	3	3.0	10700.0	93	1973	WA	7.0	1	196
Addabbo, Joseph	4	4.0	10500.0	93	1973	NY	7.0	1	196

thomas_name	Unnamed: 0	thomas_num	icpsr	congress	year	st_name	cd	dem	elec
Alexander, Bill	5	6.0	12000.0	93	1973	AR	1.0	1	1968

Statistical Summary of the data frame

```
df.describe()
```

	Unnamed: 0	thomas_num	icpsr	congress	year	cd	dem
count	10262.000000	10262.000000	10140.000000	10262.000000	10262.000000	10256.000000	1026
mean	5131.500000	5128.451618	19005.693787	104.020269	1995.040538	9.739860	0.54
std	2962.528565	2960.034117	8817.353695	6.631514	13.263028	9.850338	0.49
min	1.000000	1.000000	226.000000	93.000000	1973.000000	0.000000	0.00
25%	2566.250000	2565.250000	14264.000000	98.000000	1983.000000	3.000000	0.00
50%	5131.500000	5127.500000	15150.000000	104.000000	1995.000000	6.000000	1.00
75%	7696.750000	7691.750000	21527.250000	110.000000	2007.000000	13.000000	1.00
max	10262.000000	10254.000000	99342.000000	115.000000	2017.000000	53.000000	1.00

Slicing rows using bracket operators

```
df[1:4]
```

	Unnamed: 0	thomas_num	thomas_name	icpsr	congress	year	st_name	cd	dem
1	2	2.0	Abzug, Bella	13001.0	93	1973	NY	20.0	1
2	3	3.0	Adams, Brock	10700.0	93	1973	WA	7.0	1
3	4	4.0	Addabbo, Joseph	10500.0	93	1973	NY	7.0	1

Indexing Columns using bracket operators

```
df[['thomas_name', 'seniority']].head()
```

	thomas_name	seniority
0	Abdnor, James	1
1	Abzug, Bella	2
2	Adams, Brock	5
3	Addabbo, Joseph	7
4	Alexander, Bill	3

Passing a list of booleans to the `[]` operator

```
df_head = df.head()
row3 = [False, False, True, False, False]
df_head[row3]
```

	Unnamed: 0	thomas_num	thomas_name	icpsr	congress	year	st_name	cd	dem	elect
2	3	3.0	Adams, Brock	10700.0	93	1973	WA	7.0	1	1964

Filtering rows

```
df_head[df_head['seniority'] > 3]
```

	Unnamed: 0	thomas_num	thomas_name	icpsr	congress	year	st_name	cd	dem	el
2	3	3.0	Adams, Brock	10700.0	93	1973	WA	7.0	1	19
3	4	4.0	Addabbo, Joseph	10500.0	93	1973	NY	7.0	1	19

Filtering rows using `&` and `|` operators

Note: Each condition should be in parentheses

```
df_head[(df_head['seniority'] > 3) & (df_head['RankInParty'] < 100)]
```

	Unnamed: 0	thomas_num	thomas_name	icpsr	congress	year	st_name	cd	dem	elect
2	3	3.0	Adams, Brock	10700.0	93	1973	WA	7.0	1	1964

```
df_head[(df_head['seniority'] > 3) | (df_head['RankInParty'] < 100)]
```

	Unnamed: 0	thomas_num	thomas_name	icpsr	congress	year	st_name	cd	dem	el
2	3	3.0	Adams, Brock	10700.0	93	1973	WA	7.0	1	19
3	4	4.0	Addabbo, Joseph	10500.0	93	1973	NY	7.0	1	19
4	5	6.0	Alexander, Bill	12000.0	93	1973	AR	1.0	1	19

Filtering data using `loc` function

- `loc` is used to index/slice a group of rows and columns based on their labels.
- The first argument is the row label and the second argument is the column label.
- In the following example we index the first row and the third column.

```
df.loc[0, 'thomas_name']
```

```
'Abdnor, James'
```

```
df.loc[[0], ['thomas_name']] # now it prints as a dataframe
```

	thomas_name
0	Abdnor, James

Slicing

We can also slice rows and/or columns using the loc method. - Both the start and stop index of a slice with Ioc are inclusive. - In the following example, we slice the first 5 rows and the first 5 columns of the DataFrame. The result is a DataFrame.

```
df.loc[0:4, 'thomas_name':'st_name']
```

	thomas_name	icpsr	congress	year	st_name
0	Abdnor, James	14000.0	93	1973	SD
1	Abzug, Bella	13001.0	93	1973	NY
2	Adams, Brock	10700.0	93	1973	WA
3	Addabbo, Joseph	10500.0	93	1973	NY
4	Alexander, Bill	12000.0	93	1973	AR

Indexing and Slicing We can index and slice simultaneously as well. - In the following example we index rows and slice columns. The opposite is also possible.

```
df.loc[[3, 7], 'congress':'st_name']
```

	congress	year	st_name
3	93	1973	NY
7	93	1973	NC

Filtering data using `iloc()`

Indexing - iloc is used to index/slice a group of rows and columns. - Iloc takes row and column positions as arguments and not their labels. The first argument is the row position and the second argument is the column position. - In the following example we index the forth row and the third column. The result is a Series.

```
df.iloc[3, 2]
```

```
'Addabbo, Joseph'
```

```
df.iloc[[3], [2]] # data frame is printed
```

	thomas_name
3	Addabbo, Joseph

Slicing - We can also slice rows and/or columns using the iloc method. - We provide row and column positions for slicing using iloc. - The Start index Of a slice with iloc is inclusive. However, the end index is exclusive. - In the following example, we slice the first S rows and the first 3 columns of the DataFrame. The result is a DataFrame,

```
df.iloc[0:5, 0:3]
```

	Unnamed: 0	thomas_num	thomas_name
0	1	1.0	Abdnor, James
1	2	2.0	Abzug, Bella
2	3	3.0	Adams, Brock

	Unnamed: 0	thomas_num	thomas_name
3	4	4.0	Addabbo, Joseph
4	5	6.0	Alexander, Bill

Indexing and Slicing - We can index and slice simultaneously as well. - In the following example we index rows and slice columns. The opposite also possible.

```
df.iloc[[0, 2, 4], 0:3]
```

	Unnamed: 0	thomas_num	thomas_name
0	1	1.0	Abdnor, James
2	3	3.0	Adams, Brock
4	5	6.0	Alexander, Bill

Adding and deleting rows and columns

Adding Rows - We can add more rows to our DataFrame using the loc method. - If the row label does not exist, a new row with the specified label will be added at the end of the row.

```
df_for_add_sub = df.loc[0:5, 'congress':'st_name']
df_for_add_sub.loc[7] = [94, 1974, 'RY']
df_for_add_sub
```

	congress	year	st_name
0	93	1973	SD
1	93	1973	NY
2	93	1973	WA
3	93	1973	NY
4	93	1973	AR
5	93	1973	CA
7	94	1974	RY

Deleting Rows - We can delete rows from the DataFrame using `drop()` function by specifying `axis=0` for rows and `axis=1` for columns - Provide the labels Of the rows to be deleted as argument to the `drop()` function. - Don't forget to use `inplace=True`, otherwise the original DataFrame will remain unchanged.

```
df_for_add_sub.drop(7, axis=0, inplace=True)
```

```
df_for_add_sub
```

	congress	year	st_name
0	93	1973	SD
1	93	1973	NY

	congress	year	st_name
2	93	1973	WA
3	93	1973	NY
4	93	1973	AR
5	93	1973	CA

Adding Columns - To add a column to the DataFrame, we use the same notation as adding a key, value pair to a dictionary. - Instead Of the key, we provide column name in the square brackets, and then provide a list of values for that column. - If no column With the given name exists, a new column With the specified name and values will be added to the DataFrame.

```
df_for_add_sub['New Column'] = ['A', 'B', 'C', 'D', 'E', 'F']
df_for_add_sub
```

	congress	year	st_name	New Column
0	93	1973	SD	A
1	93	1973	NY	B
2	93	1973	WA	C
3	93	1973	NY	D
4	93	1973	AR	E
5	93	1973	CA	F

Deleting Columns - We can also delete columns of the DataFrame using drop function by specifying `axis=1` for columns. - Provide the column names to be deleted as argument to the `drop()` function. - Don't forget to use inpIace=True, otherwise the original DataFrame will remain unchanged.

```
df_for_add_sub.drop('New Column', axis=1, inplace=True)
```

```
df_for_add_sub
```

	congress	year	st_name
0	93	1973	SD
1	93	1973	NY
2	93	1973	WA
3	93	1973	NY
4	93	1973	AR
5	93	1973	CA

Sorting Values

- We can sort the values of a DataFrame with respect to a column using the `sort_values()` function, which sorts the values in ascending order by default, if you want descending order, use `ascending=False`.
- If the values of the column are alphabets, the are sorted alphabetically.

- If the values of the column are numbers, they are sorted numerically.

```
df_head.sort_values(by='seniority')
```

	Unnamed: 0	thomas_num	thomas_name	icpsr	congress	year	st_name	cd	dem
0	1	1.0	Abdnor, James	14000.0	93	1973	SD	2.0	0
1	2	2.0	Abzug, Bella	13001.0	93	1973	NY	20.0	1
4	5	6.0	Alexander, Bill	12000.0	93	1973	AR	1.0	1
2	3	3.0	Adams, Brock	10700.0	93	1973	WA	7.0	1
3	4	4.0	Addabbo, Joseph	10500.0	93	1973	NY	7.0	1

Exporting and Saving Pandas DataFrame

- To export a DataFrame as a csv file, use `to_csv()` function.
- If you do not want to store index column in the csv file, you can set `index_label=False` in the `to_csv()` function.

```
df_for_add_sub.to_csv('myfile.csv', index_label=False)
```

Concatanating DataFrames

- We can concatanate two or more dataframes below to each other by using `axis=0`
- We can concatanate two or more dataframes side-by-side to each other by using `axis=1`

```
df1 = df_for_add_sub[0:3]
df1
```

	congress	year	st_name
0	93	1973	SD
1	93	1973	NY
2	93	1973	WA

```
df2 = df_for_add_sub[4:]
df2 = df2.reset_index(drop=True)
df2
```

	congress	year	st_name
0	93	1973	AR
1	93	1973	CA

Concatanating side-by-side

```
pd.concat([df1,df2], axis=1)
```

	congress	year	st_name	congress	year	st_name
0	93	1973	SD	93.0	1973.0	AR
1	93	1973	NY	93.0	1973.0	CA
2	93	1973	WA	NaN	NaN	NaN

`groupby()` function

groupby() function is used to group DataFrame based on Series. - The DataFrame is splitted into groups. - An aggregate function is applied to each column of the splitted DataFrame. - Results are combined together. - Consider the following DataFrame.

```
data = {'Gender':['female', 'male', 'female', 'male'],'Score':[80, 83, 93, 76]}
df3 = pd.DataFrame(data)
df3
```

	Gender	Score
0	female	80
1	male	83
2	female	93
3	male	76

```
df3.groupby(df3['Gender']).mean()
```

Gender	Score
female	86.5
male	79.5

17.2 Data Analysis in The Yelp Dataset

Information about local businesses in 13 cities in PA and NV “yelp_data” tab data columns:

- name: Name of business
- category _0: ISt user-assigned business category
- category_l: 2nd user-assigned business category
- take-out: Flag (True/FaIse) indicating if business provides take-out
- review count: Number of reviews
- stars: Overall star rating
- city_id: Identifier referencing city of business (match to id on “cities” tab)
- state_id: Identifier referencing state of business (match to id on “states” tab)

“cities” tab data columns: - id: Unique identifier of city - city: City name

“states” tab data columns: - id: Unique identifier of state - state: State name

17.2.1 Loading Data

```
#yelp_df = pd.read_excel('yelp.xlsx') this could have been done but the excel file has
    multiple sheets
# we gonna read each sheet individually

data_file = pd.ExcelFile('yelp.xlsx') # this is the whole excel file
yelp_data = data_file.parse( 'yelp_data') # read the 'yelp_data' sheet
```

```
yelp_data.shape # gives dimension of the data
```

```
(600, 8)
```

17.2.2 Inspecting Data

```
yelp_data.count() # Get a count of values in each column
```

```
name            600
category_0      600
category_1      600
take_out        600
review_count    600
stars           600
city_id         600
state_id        600
dtype: int64
```

```
print(yelp_data.columns) # displays column names
print(yelp_data.dtypes) # type of data in each column
```

```
Index(['name', 'category_0', 'category_1', 'take_out', 'review_count', 'stars',
       'city_id', 'state_id'],
      dtype='object')
name             object
category_0       object
category_1       object
take_out           bool
review_count      int64
stars           float64
city_id           int64
state_id          int64
dtype: object
```

```
yelp_data.describe() # summary statistics
```

	review_count	stars	city_id	state_id
count	600.000000	600.000000	600.000000	600.000000
mean	33.771667	3.495000	9.193333	1.500000
std	86.901895	0.955596	2.997933	0.500417
min	3.000000	1.000000	1.000000	1.000000
25%	5.000000	3.000000	8.000000	1.000000
50%	10.000000	3.500000	10.500000	1.500000
75%	25.250000	4.000000	12.000000	2.000000
max	1305.000000	5.000000	13.000000	2.000000

review_count	stars	city_id	state_id

```
yelp_data.head()
```

	name	category_0	category_1	take_out	review_count	stars	ci
0	China Sea Chinese Restaurant	Restaurants	Chinese	True	11	2.5	1
1	Discount Tire Center	Tires	Automotive	False	24	4.5	1
2	Frankfurters	Restaurants	Hot Dogs	True	3	4.5	1
3	Fred Dietz Floral	Shopping	Flowers & Gifts	False	6	4.0	1
4	Kuhn's Market	Food	Grocery	False	8	3.5	1

```
yelp_data = yelp_data.drop_duplicates()
```

17.2.2.1 Joining data

```
yelp_cities = data_file.parse('cities') # the `cities` sheet
```

```
yelp_cities.head()
```

	id	city
0	1	Bellevue
1	2	Braddock
2	3	Carnegie
3	4	Homestead
4	5	Mc Kees Rocks

```
yelp_df = pd.merge(left=yelp_data, right=yelp_cities, how='inner', left_on='city_id',
    right_on='id')
# the left and right on's are the columns to be matched
```

```
yelp_df.head()
```

	name	category_0	category_1	take_out	review_count	stars	ci
0	China Sea Chinese Restaurant	Restaurants	Chinese	True	11	2.5	1
1	Discount Tire Center	Tires	Automotive	False	24	4.5	1
2	Frankfurters	Restaurants	Hot Dogs	True	3	4.5	1
3	Fred Dietz Floral	Shopping	Flowers & Gifts	False	6	4.0	1
4	Kuhn's Market	Food	Grocery	False	8	3.5	1

```
yelp_states = data_file.parse('states')
```

```
yelp_states.head()
```

	id	state
0	1	PA
1	2	NV

```
yelp_df = pd.merge(left=yelp_df, right=yelp_states, how='inner', left_on='state_id', right_on
    ='id')
```

```
yelp_df.shape
```

```
(600, 12)
```

```
yelp_df.head()
```

	name	category_0	category_1	take_out	review_count	stars	ci
0	China Sea Chinese Restaurant	Restaurants	Chinese	True	11	2.5	1
1	Discount Tire Center	Tires	Automotive	False	24	4.5	1
2	Frankfurters	Restaurants	Hot Dogs	True	3	4.5	1
3	Fred Dietz Floral	Shopping	Flowers & Gifts	False	6	4.0	1
4	Kuhn's Market	Food	Grocery	False	8	3.5	1

17.2.3 Quering Data

We want to see name, city and state of first 5 businesses

```
yelp_df[['name', 'city', 'state']].head(5)
```

	name	city	state
0	China Sea Chinese Restaurant	Bellevue	PA
1	Discount Tire Center	Bellevue	PA
2	Frankfurters	Bellevue	PA
3	Fred Dietz Floral	Bellevue	PA
4	Kuhn's Market	Bellevue	PA

```
# delete unnecessary additional columns
del yelp_df['id_x']
del yelp_df['id_y']
```

```
yelp_df.head()
```

	name	category_0	category_1	take_out	review_count	stars	ci
0	China Sea Chinese Restaurant	Restaurants	Chinese	True	11	2.5	1
1	Discount Tire Center	Tires	Automotive	False	24	4.5	1
2	Frankfurters	Restaurants	Hot Dogs	True	3	4.5	1
3	Fred Dietz Floral	Shopping	Flowers & Gifts	False	6	4.0	1
4	Kuhn's Market	Food	Grocery	False	8	3.5	1

17.2.3.1 Quering Data - Slicing Rows

Format: `[start(inclusive), end(exclusive)]`

```
yelp_df[-1:]['name'] # returns the object in the 'name' column of the last row
```

```
599    A Sunrise Towing
Name: name, dtype: object
```

17.2.4 Querying Data - Conditions Using Boolean Indexing

Select the businessess in Pittsburgh

```
yelp_pitts = yelp_df[yelp_df['city'] == 'Pittsburgh']
yelp_pitts.head()
```

	name	category_0	category_1	ta
95	Aamco Transmissions	Auto Repair	Automotive	Fa
96	Animal Rescue League Shelter & Wildlife Center	Animal Shelters	Veterinarians	Fa
97	Aracri's Greentree Inn	Italian	American (New)	T
98	Atch-Mont Real Estate	Real Estate Services	Property Management	Fa
99	Atria's Restaurant	American (New)	Sandwiches	T

Select the Bars

```
yelp_df[(yelp_df['category_0'] == 'Bars') | (yelp_df['category_1'] == 'Bars')].head()
```

	name	category_0	category_1	take_out	review_count	stars
12	Emil's Lounge	Bars	American (New)	True	26	4.5
15	Alexion's Bar & Grill	Bars	American (Traditional)	True	23	4.0
32	Rocky's Lounge	Bars	American (Traditional)	True	10	4.0
42	Duke's Upper Deck Cafe	Pubs	Bars	True	33	3.5
62	Randy's Beer Barrel Pub	Pubs	Bars	False	3	2.5

Select the bars in Carnegie

```
cat_0_bars = yelp_df['category_0'] == 'Bars'
cat_1_bars = yelp_df['category_1'] == 'Bars'
carnegie = yelp_df['city'] == 'Carnegie'
yelp_df[(cat_0_bars | cat_1_bars) & carnegie]
```

	name	category_0	category_1	take_out	review_count	stars	city
15	Alexion's Bar & Grill	Bars	American (Traditional)	True	23	4.0	3
32	Rocky's Lounge	Bars	American (Traditional)	True	10	4.0	3

```
# Alternatively
yelp_df[((yelp_df['category_0'] == 'Bars') | (yelp_df['category_1'] == 'Bars')) & (yelp_df['
    city'] == 'Carnegie')]
```

	name	category_0	category_1	take_out	review_count	stars	city
15	Alexion's Bar & Grill	Bars	American (Traditional)	True	23	4.0	3
32	Rocky's Lounge	Bars	American (Traditional)	True	10	4.0	3

Select the bars and restaurants in Carnegie

```
cat_0 = yelp_df['category_0'].isin(['Bars', 'Restaurants']) # tests if category_0 is in the
    provided list
cat_1 = yelp_df['category_1'].isin(['Bars', 'Restaurants']) # tests if category_1 is in the
    provided list
yelp_df[(cat_0 | cat_1) & carnegie]
```

	name	category_0	category_1	take_out	review_count	s
15	Alexion's Bar & Grill	Bars	American (Traditional)	True	23	4
18	Barb's Country Junction Cafe	Restaurants	Cafes	True	9	4
20	Don Don Chinese Restaurant	Restaurants	Chinese	True	10	2
29	Papa J's	Restaurants	Italian	True	81	3
30	Porto Fino Pizzaria & Gyro	Restaurants	Pizza	False	4	2
32	Rocky's Lounge	Bars	American (Traditional)	True	10	4

How many total Dive bars are there in Las Vegas

```
# Look for Dive Bars in the data frame
cat_0_db = yelp_df['category_0'] == 'Dive Bars'
cat_1_db = yelp_df['category_1'] == 'Dive Bars'

# limit it to Las Vegas
lv = yelp_df['city'] == 'Las Vegas'

# combine
db_lv = yelp_df[(cat_0_db | cat_1_db) & lv]

# Print the results
print('There are', len(db_lv), 'Dive Bars in Las Vegas')
```

```
There are 3 Dive Bars in Las Vegas
```

Recommend a random dive bar with at least a 4 star rating

- Look at the total set of dive bars above and query for those that have a star rating of at least 4.0
- Import the random module: import random
- Get a random number using the randint method
- Get a random dive bar from the set above using the random number

```
stars = db_lv['stars'] >= 4.0
db_lv_4rating = db_lv[stars]

import random
# get random number between 0 and last index
rand_int = random.randint(0, len(db_lv_4rating) - 1)
# get random dive bar based on random number
```

```
rand_db = db_lv_4rating[rand_int: rand_int + 1]
rand_db
```

	name	category_0	category_1	take_out	review_count	stars	city_id	state_id
451	Huntridge Tavern	Dive Bars	Bars	False	50	4.0	12	2

Calculate the total number of reviews for nail salons in Henderson

```
cat_0 = yelp_df['category_0'].str.contains('Nail Salon') # tests if category_0 string value
    contains Nail Salon
cat_1 = yelp_df['category_1'].str.contains('Nail Salon') # tests if category_1 string value
    contains Nail Salon
henderson = yelp_df['city'] == 'Henderson'
yelp_df[(cat_0 | cat_1) & henderson]['review_count'].sum()
```

```
158
```

Calculate the average star rating for auto repair shops in Pittsburgh

```
cat_0 = yelp_df['category_0'].str.contains('Auto Repair')
cat_1 = yelp_df['category_0'].str.contains('Auto Repair')
pitts = yelp_df['city'] == 'Pittsburgh'
yelp_df[(cat_0 | cat_1) & pitts]['stars'].mean()
```

```
4.5
```

What cities are in the yelp dataset

```
yelp_df['city'].unique() # Returns unique values in city column
```

```
array(['Bellevue', 'Braddock', 'Carnegie', 'Homestead', 'Mc Kees Rocks',
       'Mount Lebanon', 'Munhall', 'Pittsburgh', 'West Homestead',
       'West Mifflin', 'Henderson', 'Las Vegas', 'North Las Vegas'],
      dtype=object)
```

How many businesses are in each city

```
yelp_df['city'].value_counts()
```

```
Pittsburgh         193
Las Vegas          133
Henderson          130
Homestead           41
North Las Vegas     37
Carnegie            22
Bellevue            12
Mc Kees Rocks       10
West Mifflin         9
Mount Lebanon        4
Munhall              4
West Homestead       3
Braddock             2
Name: city, dtype: int64
```

How many unique user assigned business categories are ther in category_0

```
yelp_df['category_0'].nunique()
```

```
89
```

17.2.4.1 Updating and Creating Data

Add a new "categories" column that combines "category_0" and "category_1" as a commaseparated list

```
yelp_df['categories'] = yelp_df['category_0'].str.cat(yelp_df['category_1'], sep=',') #
    concatenates the string value of category_0 with category_1, separated by a comma ie ','
```

Now we can look up businesses based on the single "category" column

```
yelp_df[yelp_df['categories'].str.contains('Pizza')].head()
```

	name	category_0	category_1	take_out	review_count	stars	city_id
6	Luigi's Pizzeria	Restaurants	Pizza	True	18	4.0	1
8	R & B's Pizza Place	Restaurants	Pizza	True	17	4.0	1
30	Porto Fino Pizzaria & Gyro	Restaurants	Pizza	False	4	2.5	3
48	Homestead Capri Pizza	Italian	Pizza	True	4	2.0	4
49	Italian Village Pizza	Restaurants	Pizza	False	6	2.5	4

- Add a new "rating" column that converts "stars" to a comparable value in the 10-point system

```
yelp_df['rating'] = yelp_df['stars'] * 2
```

- Now, update the new "rating" column so that it displays the rating as "x out of 10" First, create a helper function that will take a rating value as an argument and concatenate a string to it

```
def convert_to_string(x):
    return (str(x) + 'out of 10') # casts x (rating) to a string, then concatenates another
    string
```

use the `apply()` method to run the helper function for the rating in each row

```
yelp_df['rating'] = yelp_df['rating'].apply(convert_to_string) # applies function
yelp_df[['name', 'review_count', 'rating']].head()
```

	name	review_count	rating
0	China Sea Chinese Restaurant	11	5.0out of 10
1	Discount Tire Center	24	9.0out of 10
2	Frankfurters	3	9.0out of 10
3	Fred Dietz Floral	6	8.0out of 10
4	Kuhn's Market	8	7.0out of 10

17.2.4.2 Querying Data – agg()

Let's find out the sum, mean, and standard deviation for the star ratings of each city

```
import numpy as np
yelp_df.groupby(['city']).agg([np.sum, np.mean, np.std])['stars']
```

city	sum	mean	std
Bellevue	45.0	3.750000	0.783349
Braddock	9.5	4.750000	0.353553
Carnegie	76.0	3.454545	0.688495
Henderson	444.5	3.419231	0.906060
Homestead	134.5	3.280488	0.837024
Las Vegas	452.0	3.398496	1.042214
Mc Kees Rocks	37.0	3.700000	0.856349
Mount Lebanon	12.5	3.125000	1.108678
Munhall	12.0	3.000000	0.816497
North Las Vegas	112.0	3.027027	1.073325
Pittsburgh	718.0	3.720207	0.902517
West Homestead	10.0	3.333333	1.040833
West Mifflin	34.0	3.777778	1.227577

17.3 Pivot Tables

A pivot table is a useful data summarization tool that creates a new table from the contents in the DataFrame.

✷ Note: By default, the pivot table calculates average (mean) for each column

```
pv_city = pd.pivot_table(yelp_df, index=['city'])
pv_city
```

city	city_id	review_count	stars	state_id	take_out
Bellevue	1	13.166667	3.750000	1	0.500000
Braddock	2	14.500000	4.750000	1	0.500000
Carnegie	3	13.590909	3.454545	1	0.409091
Henderson	11	33.323077	3.419231	2	0.238462
Homestead	4	23.243902	3.280488	1	0.268293
Las Vegas	12	54.330827	3.398496	2	0.218045
Mc Kees Rocks	5	10.700000	3.700000	1	0.700000
Mount Lebanon	6	6.250000	3.125000	1	0.250000
Munhall	7	22.750000	3.000000	1	0.750000
North Las Vegas	13	10.756757	3.027027	2	0.216216
Pittsburgh	8	33.523316	3.720207	1	0.430052
West Homestead	9	41.333333	3.333333	1	0.666667
West Mifflin	10	5.666667	3.777778	1	0.333333

It is possible to use more than one index

```
pv_st_tk = pd.pivot_table(yelp_df, index=['state', 'take_out'])
pv_st_tk
```

state	take_out	city_id	review_count	stars	state_id
NV	False	11.698276	16.900862	3.409483	2
	True	11.661765	118.161765	3.198529	2
PA	False	6.643678	11.580460	3.695402	1
	True	6.769841	49.936508	3.535714	1

- Create a pivot table that displays the average (mean) review count and star rating for bars and restaurants in each city

```
# filtering bars and restaurant
ba_res = yelp_df['category_0'].isin(['Bars', 'Restaurants'])
# creating the filtered dataframe
df_ba_res = yelp_df[ba_res]
pv_st_ct_cat0_ba_res = pd.pivot_table(df_ba_res, index=['state', 'city', 'category_0'])

# filter only 'review_count' and 'stars'
pv_st_ct_cat0_ba_res[['review_count', 'stars']]
```

state	city	category_0	review_count	stars
NV	Henderson	Bars	171.000000	3.000000
		Restaurants	102.454545	3.181818
	Las Vegas	Bars	15.500000	4.000000
		Restaurants	221.153846	3.153846
	North Las Vegas	Bars	7.000000	3.500000
		Restaurants	12.000000	3.000000
PA	Bellevue	Restaurants	14.000000	3.916667
	Braddock	Bars	26.000000	4.500000
	Carnegie	Bars	16.500000	4.000000
		Restaurants	26.000000	3.125000
	Homestead	Bars	23.000000	2.500000
		Restaurants	6.000000	2.500000
	Mc Kees Rocks	Bars	9.000000	3.500000
		Restaurants	7.333333	3.333333
	Munhall	Restaurants	9.500000	3.500000
	Pittsburgh	Bars	20.000000	3.416667
		Restaurants	67.000000	3.203704
	West Homestead	Bars	92.000000	2.500000
	West Mifflin	Restaurants	5.000000	4.333333

17.3.0.1 Pivot Tables – `aggfunc()`

- To display summary statistics other than the average (mean)
  - Use the `aggfunc` parameter to specify the aggregation function(s)
  - Use the `values` parameter to specify the column(s) for the aggfunc

In our dataset, how many (sum) reviews does each city have?

```
import numpy as np
```

```
pv_agg = pd.pivot_table(
    yelp_df, index=['state', 'city'],
    values=['review_count'], # specify the column(s) for the aggfunc
    aggfunc=[np.sum] # specify the aggregation function(s)
)
pv_agg
```

		sum
		review_count
state	city	
NV	Henderson	4332
	Las Vegas	7226
	North Las Vegas	398
PA	Bellevue	158
	Braddock	29
	Carnegie	299
	Homestead	953
	Mc Kees Rocks	107
	Mount Lebanon	25
	Munhall	91
	Pittsburgh	6470
	West Homestead	124
	West Mifflin	51

It's possible to further segment our results using the "columns" parameter

```
pv_agg2 = pd.pivot_table(
    yelp_df, index=['state', 'city'],
    values=['review_count'], # specify the column(s) for the aggfunc
    columns=['take_out'], # specify the columns to separate the results
    aggfunc=[np.sum] # specify the aggregation function(s)
)
pv_agg2
```

		sum	
		review_count	
	take_out	False	True
state	city		
NV	Henderson	2009	2323
	Las Vegas	1619	5607
	North Las Vegas	293	105
PA	Bellevue	52	106
	Braddock	3	26
	Carnegie	74	225
	Homestead	323	630
	Mc Kees Rocks	48	59

		sum review_count	
	take_out	False	True
state	city		
	Mount Lebanon	13	12
	Munhall	12	79
	Pittsburgh	1447	5023
	West Homestead	7	117
	West Mifflin	36	15

- We can also pass as an argument to aggfunc(), a dict object containing different aggregate functions to perform on different values
- If we want to see the total number of review counts, their mean, standard deviation and skewness and average ratings

```
from scipy.stats import skew
```

```
pv_agg3 = pd.pivot_table(
    yelp_df, index=['state', 'city'],
    columns=['take_out'],
    aggfunc={'review_count':[np.sum, np.mean, np.std, skew], 'stars': np.mean}
)
pv_agg3
```

		review_count						
		mean		skew		std		sum
	take_out	False	True	False	True	False	True	False
state	city							
NV	Henderson	20.292929	74.935484	2.494535	1.748795	25.752849	87.888541	2009
	Las Vegas	15.567308	193.344828	3.656274	2.171055	20.805509	305.640344	1619
	North Las Vegas	10.103448	13.125000	1.827811	1.229328	8.001539	8.253787	293
PA	Bellevue	8.666667	17.666667	1.559679	-0.023273	7.737355	10.191500	52
	Braddock	3.000000	26.000000	NaN	NaN	NaN	NaN	3
	Carnegie	5.692308	25.000000	0.612741	1.279319	3.010665	28.930952	74
	Homestead	10.766667	57.272727	1.919527	0.561612	10.122298	51.423907	323
	Mc Kees Rocks	16.000000	8.428571	0.705411	1.080011	21.656408	6.579188	48
	Mount Lebanon	4.333333	12.000000	0.381802	NaN	1.527525	NaN	13
	Munhall	12.000000	26.333333	NaN	0.683954	NaN	29.263174	12
	Pittsburgh	13.154545	60.518072	3.195666	2.541720	17.726353	74.987050	1447
	West Homestead	7.000000	58.500000	NaN	0.000000	NaN	47.376154	7
	West Mifflin	6.000000	5.000000	1.718385	-0.707107	5.932959	1.732051	36

Chapter 18

Matplotlib

- ✓ Used to build plots
- ✓ 2D, 3D plots and animations
- ✓ Visualization in details on Data Visualization with Python Book.

```
import matplotlib
```

```
matplotlib.__version__
```

```
'3.5.2'
```

18.1 Line Plots

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

The code `%matplotlib inline` above is to make plots show in the notebook output.

```
x = np.arange(0, 4 * np.pi, 0.1)
y = np.sin(x)
fig = plt.figure(figsize=(6,3))
plt.plot(x, y, linewidth=1)
plt.show()
```

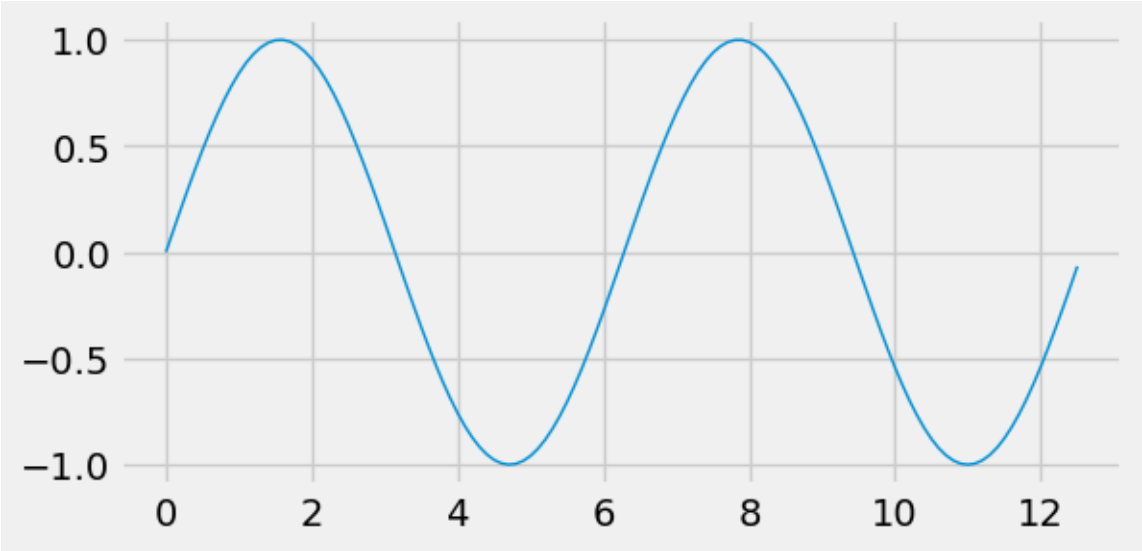


Figure 18.1: A Basic line plot

18.2 Saving Plots

```
x = [0, 2, 4, 6]
y = [1, 2, 4, 8]
fig = plt.figure(figsize=(6,3))
plt.plot(x, y, linewidth=1)
plt.xlabel('x values')
plt.ylabel('y values')
plt.title('Plotted x and y values')
plt.legend(['Data 1'])

# Saving the plot
plt.savefig('plot1.png', dpi=350, bbox_inches='tight')

plt.show()
```

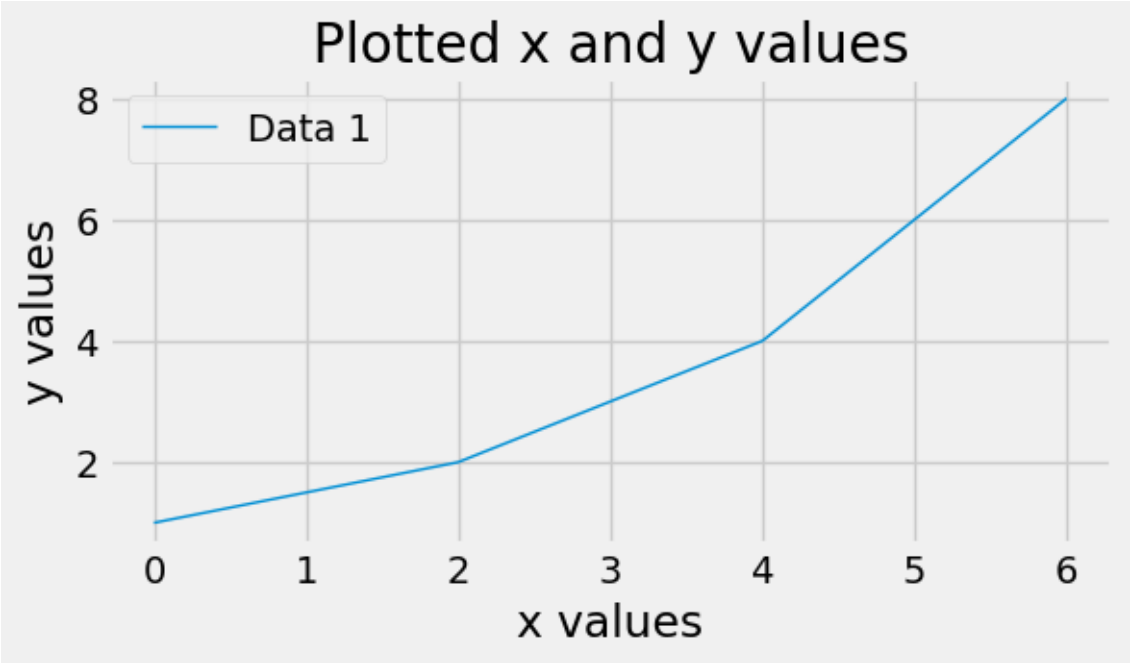


Figure 18.2: Saved plot

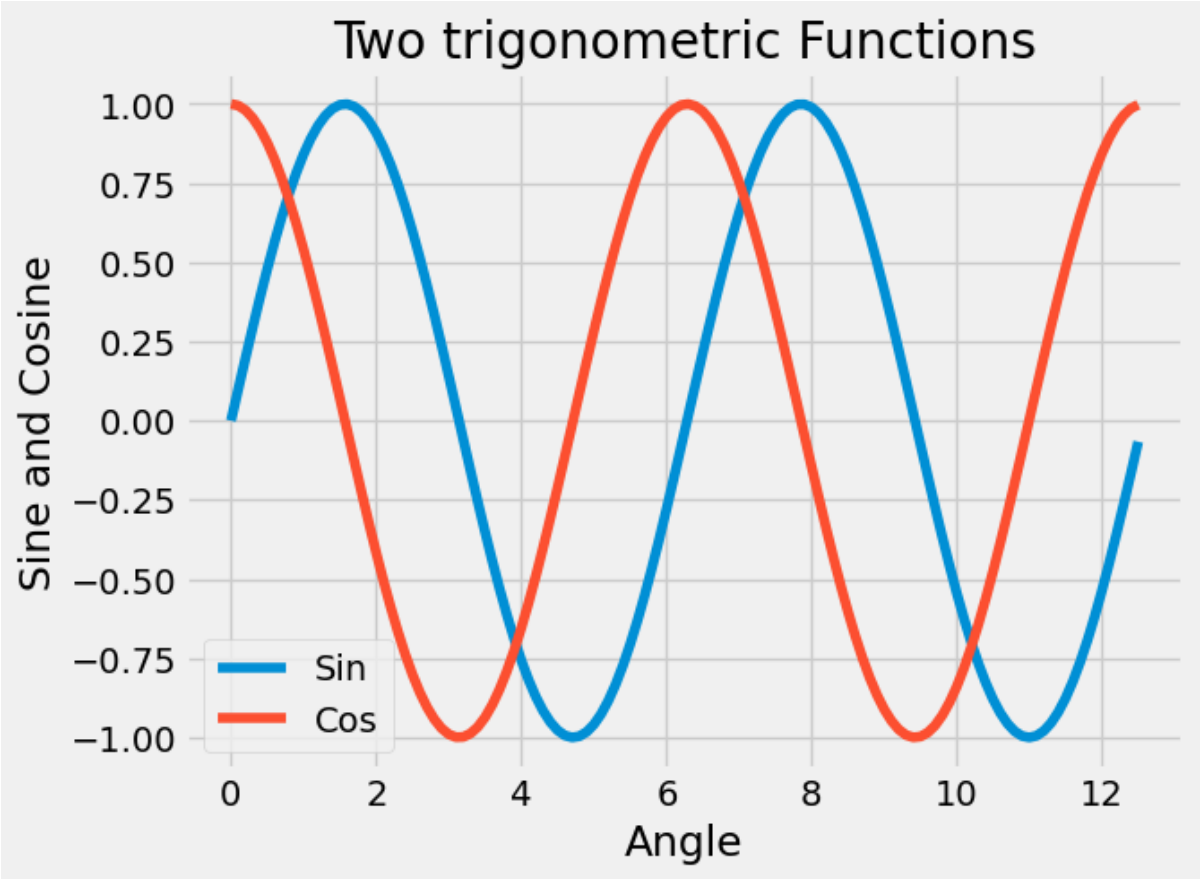
18.3 Multi Line plots

- ✓ Multi Line Plots
- ✓ Object-Oriented Interface

```
x = np.arange(0, 4 * np.pi, 0.1)
y = np.sin(x)
z = np.cos(x)

fig = plt.figure(figsize=(6,3))
fig, ax = plt.subplots()

ax.plot(x, y)
ax.plot(x, z)

ax.xaxis.set_label_text('Angle')
ax.yaxis.set_label_text('Sine and Cosine')
ax.legend(['Sin', 'Cos'])
ax.set_title('Two trigonometric Functions')

plt.show()
```

```
<Figure size 600x300 with 0 Axes>
```

Multi plots



18.4 Histogram

```
plt.style.use('fivethirtyeight')
np.random.seed(1234)
x = np.random.chisquare(10, 100)

fig, ax = plt.subplots()

ax.hist(x, 30)
ax.set_xlabel('Bin range')
ax.set_ylabel('Frequency')
ax.set_title('Histogram')

fig.tight_layout()
plt.show()
```

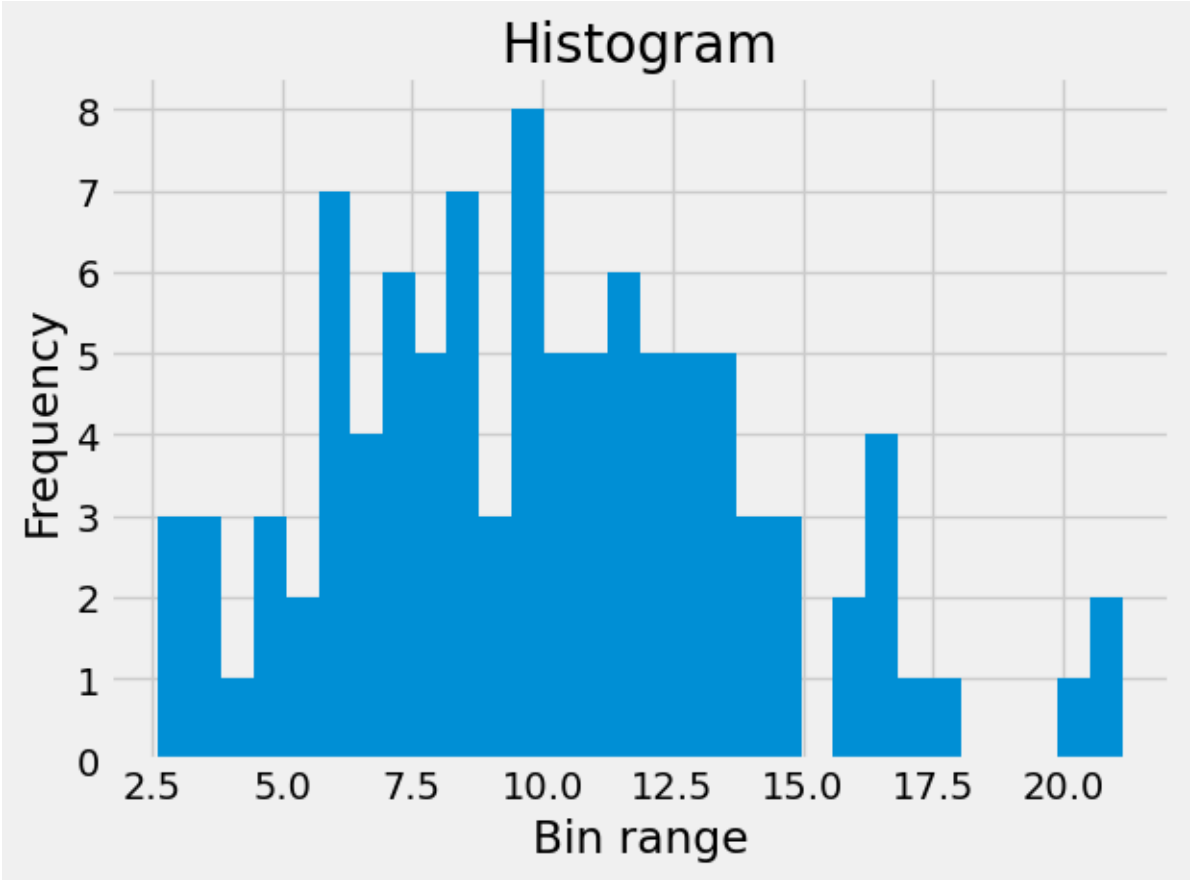


Figure 18.3: Histogram

Chapter 19

Seaborn Library For Data Science

- Seaborn is another visualization Python library built on top of Matplotlib.
- It extends the functionality of Matplotlib and allows creating a variety of different graphs with fewer syntax.
- More details can be found on Data Visualization with Python