

# Data Science & Machine Learning

Eman Raslan  



# Python Basics

# Agenda

- Data types
- Arithmetic operations
- Assignment operators
- Input Function
- Print formatting
- Flow Control
- Comparison and Logical operators
- Range Function
- Break, continue and pass
- Nested Statements and Scope
- Functions
- classes
- Reading, writing and append to files
- Errors and Exception Handling
- Lambda, Map, and filter functions
- List comprehension
- NumPy module
- Pandas module

# Data Types

Name	Type	Description	Example
Integers	Int	Whole numbers	3 300 200 -2
Floats	Float	Numbers with decimal point	2.3 4.0 -5.3
Strings	Str	Ordered sequence of characters	"hello" 'Eman' "2000"
Lists	List	Ordered sequence of objects	[10,"omar",2.33]
Dictionaries	dict	Unordered Key:Value pairs	{"key1":"value1", "key2":"value2"}
Tuples	Tup	Ordered immutable sequence of objects	("hello",4.4,200)
Sets	Set	Unordered collection of unique objects	{"a",3000}
Booleans	bool	Logical value indicating True or False	True False

# Arithmetic Operators

- + (Addition)
- (Subtraction)
- \* (Multiplication)
- / (Division)
- // (Floor division)
- % (Modulus)
- \*\* (Exponentiation)

# Variable declaration

## Rules for variable names

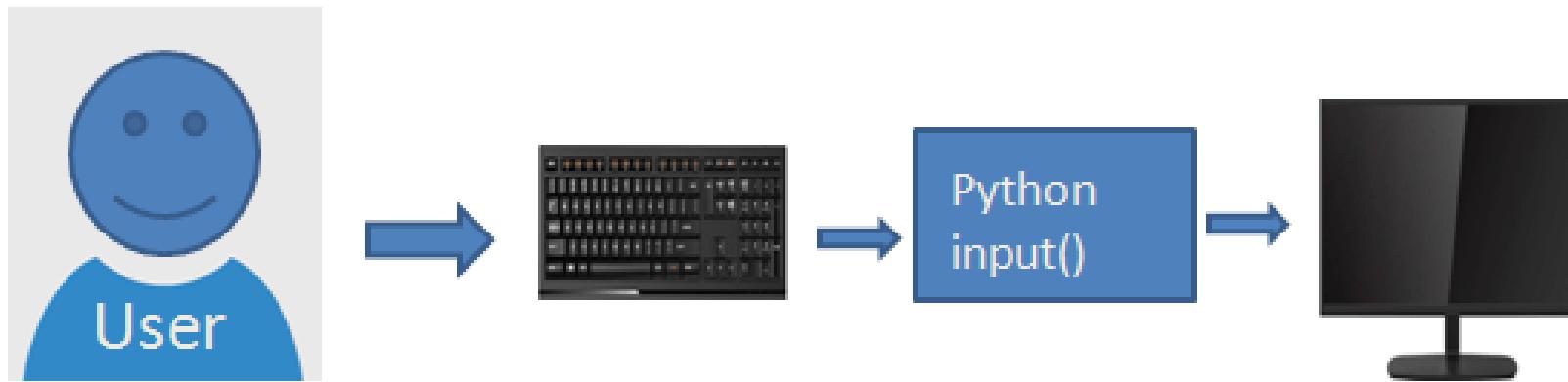
- names can not start with a **number**
- names can not contain **spaces**, use `_` instead
- names can not contain any of these symbols:  
`:'"<>/?|\\!@#%^&*~-+`
- avoid using Python built-in **keywords** like `'list'` and `'str'`
- avoid using the single characters `'l'` (lowercase letter el), `'O'` (uppercase letter oh) and `'I'` (uppercase letter eye) as they can be confused with `'1'` and `'0'`.

# Assignment operators

Operator	Meaning	Equivalent
= (Assign)	a=5 Assign 5 to variable a	a = 5
+= (Add and assign)	a+=5 Add 5 to a and assign it as a new value to a	a = a+5
-= (Subtract and assign)	a-=5 Subtract 5 from variable a and assign it as a new value to a	a = a-5
*= (Multiply and assign)	a*=5 Multiply variable a by 5 and assign it as a new value to a	a = a*5
/= (Divide and assign)	a/=5 Divide variable a by 5 and assign a new value to a	a = a/5
%= (Modulus and assign)	a%=5 Performs modulus on two values and assigns it as a new value to a	a = a%5
**= (Exponentiation and assign)	a**=5 Multiply a five times and assigns the result to a	a = a**5
//= (Floor-divide and assign)	a//=5 Floor-divide a by 5 and assigns the result to a	a = a//5

# Input

## Python Input() function



# Strings

**STRING = "AASHINA"**

**REVERSE INDEX**

-7	-6	-5	-4	-3	-2	-1
0	1	2	3	4	5	6
A	A	S	H	I	N	A

**FORWARD INDEX**

**STRING[0] = 'A'**

**STRING[1] = 'A'**

**STRING[2] = 'S'**

**STRING[3] = 'H'**

**STRING[4] = 'I'**

**STRING[5] = 'N'**

**STRING[6] = 'A'**

**STRING[-7] = 'A'**

**STRING[-6] = 'A'**

**STRING[-5] = 'S'**

**STRING[-4] = 'H'**

**STRING[-3] = 'I'**

**STRING[-2] = 'N'**

**STRING[-1] = 'A'**

# Lists

```
L = [ 20, 'Jessa', 35.75, [30, 60, 90] ]  
      ↑   ↑   ↑   ↑  
L[0] L[1] L[2] L[3]
```

- ✓ **Ordered:** Maintain the order of the data insertion.
- ✓ **Changeable:** List is mutable and we can modify items.
- ✓ **Heterogeneous:** List can contain data of different types
- ✓ **Contains duplicate:** Allows duplicates data

# List Operations

Operation	Description
<code>x in l1</code>	Check if the list <code>l1</code> contains item <code>x</code> .
<code>x not in l2</code>	Check if list <code>l1</code> does not contain item <code>x</code> .
<code>l1 + l2</code>	Concatenate the lists <code>l1</code> and <code>l2</code> . Creates a new list containing the items from <code>l1</code> and <code>l2</code> .
<code>l1 * 5</code>	Repeat the list <code>l1</code> 5 times.
<code>l1[i]</code>	Get the item at index <code>i</code> .
<code>l1[i:j]</code>	List slicing. Get the items from index <code>i</code> up to index <code>j</code> (excluding <code>j</code> ) as a List.
<code>l1[i:j:k]</code>	List slicing with step. Returns a List with the items from index <code>i</code> up to index <code>j</code> taking every <code>k</code> -th item.
<code>len(l1)</code>	Returns a count of total items in a list.
<code>l2.count(60)</code>	Returns the number of times a particular item (60) appears in a list..

# List Operations

Operation	Description
l1.index(30)	Returns the index number of a particular item (30) in a list.
l1.index(30, 2, 5)	Returns the index number of a particular item (30) in a list. But search Returns the item with maximum value from a list.
min(l1)	Returns the item with a minimum value from a list.
max(l1)	Returns the item with maximum value from a list.
l1.append(100)	Add item at the end of the list
l1.append([2, 5, 7])	Append the nested list at the end
l1[2] = 40	Modify the item present at index 2
l1.remove(40)	Removes the first occurrence of item 40 from the list.
pop(2)	Removes and returns the item at index 2 from the list.
l1.clear()	Make list empty
l3= l1.copy()	Copy l1 into l2

# Tuples

$$T = ( 20, \text{'Jessa'}, 35.75, [30, 60, 90] )$$

↑      ↑      ↑      ↑  
T[0]    T[1]    T[2]    T[3]

- ✓ **Ordered:** Maintain the order of the data insertion.
- ✓ **Unchangeable:** Tuples are immutable and we can't modify items.
- ✓ **Heterogeneous:** Tuples can contain data of types
- ✓ **Contains duplicate:** Allows duplicates data

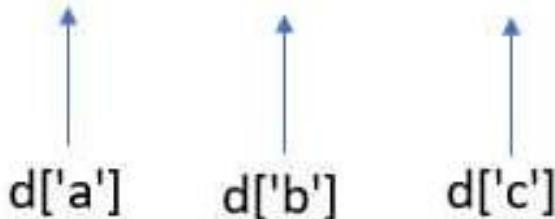
# Tuples operations

Operation	Description
x in t1	Check if the tuple t1 contains the item x.
x not in t2	Check if the tuple t1 does not contain the item x.
t1 + t2	Concatenate the tuples t1 and t2. Creates a new tuple containing the items from t1 and t2.
t1 * 5	Repeat the tuple t1 5 times.
t1[i]	Get the item at the index i.
t1[i:j]	Tuple slicing. Get the items from index i up to index j (excluding j) as a tuple.
t1[i:j:k]	Tuple slicing with step. Return a tuple with the items from index i up to index j taking every k-th item.
len(t1)	Returns a count of total items in a tuple
t2.count(60)	Returns the number of times a particular item (60) appears in a tuple.
t1.index(30)	Returns the index number of a particular item(30) in a tuple.
t1.index(40, 2, 5)	Returns the index number of a particular item(30) in a tuple. But search only from index number 2 to 5.
min(t1)	Returns the item with a minimum value from a tuple
max(t1)	Returns the item with maximum value from a tuple

# Dictionary

Unordered collections of unique values stored in (Key-Value) pairs.

```
d = {'a': 10, 'b': 20, 'c': 30}
```



- ✓ **Unordered:** The items in dict are stored without any index value
- ✓ **Unique:** Keys in dictionaries should be Unique
- ✓ **Mutable:** We can add/Modify/Remove key-value after the creation

# Dictionary Operations

Operations	Description
dict({'a': 10, 'b': 20})	Create a dictionary using a dict() constructor.
d2 = {}	Create an empty dictionary.
d1.get('a')	Retrieve value using the key name a.
d1.keys()	Returns a list of keys present in the dictionary.
d1.values()	Returns a list with all the values in the dictionary.
d1.items()	Returns a list of all the items in the dictionary with each key-value pair inside a tuple.
len(d1)	Returns number of items in a dictionary.
d1['d'] = 40	Update dictionary by adding a new key.
d1.update({'e': 50, 'f': 60})	Add multiple keys to the dictionary.
d1.setdefault('g', 70)	Set the default value if a key doesn't exist.
d1['b'] = 100	Modify the values of the existing key.
d1.pop('b')	Remove the key b from the dictionary.
d1.popitem()	Remove any random item from a dictionary.
d1.clear()	Removes all items from the dictionary.
'key' in d1.keys()	Check if a key exists in a dictionary.
d1.update(d2)	Add all items of dictionary d2 into d1.
d3= **d1, **d2}	Join two dictionaries.
d2 = d1.copy()	Copy dictionary d1 into d2.
max(d1)	Returns the key with the maximum value in the dictionary d1
min(d1)	Returns the key with the minimum value in the dictionary d1

# Set

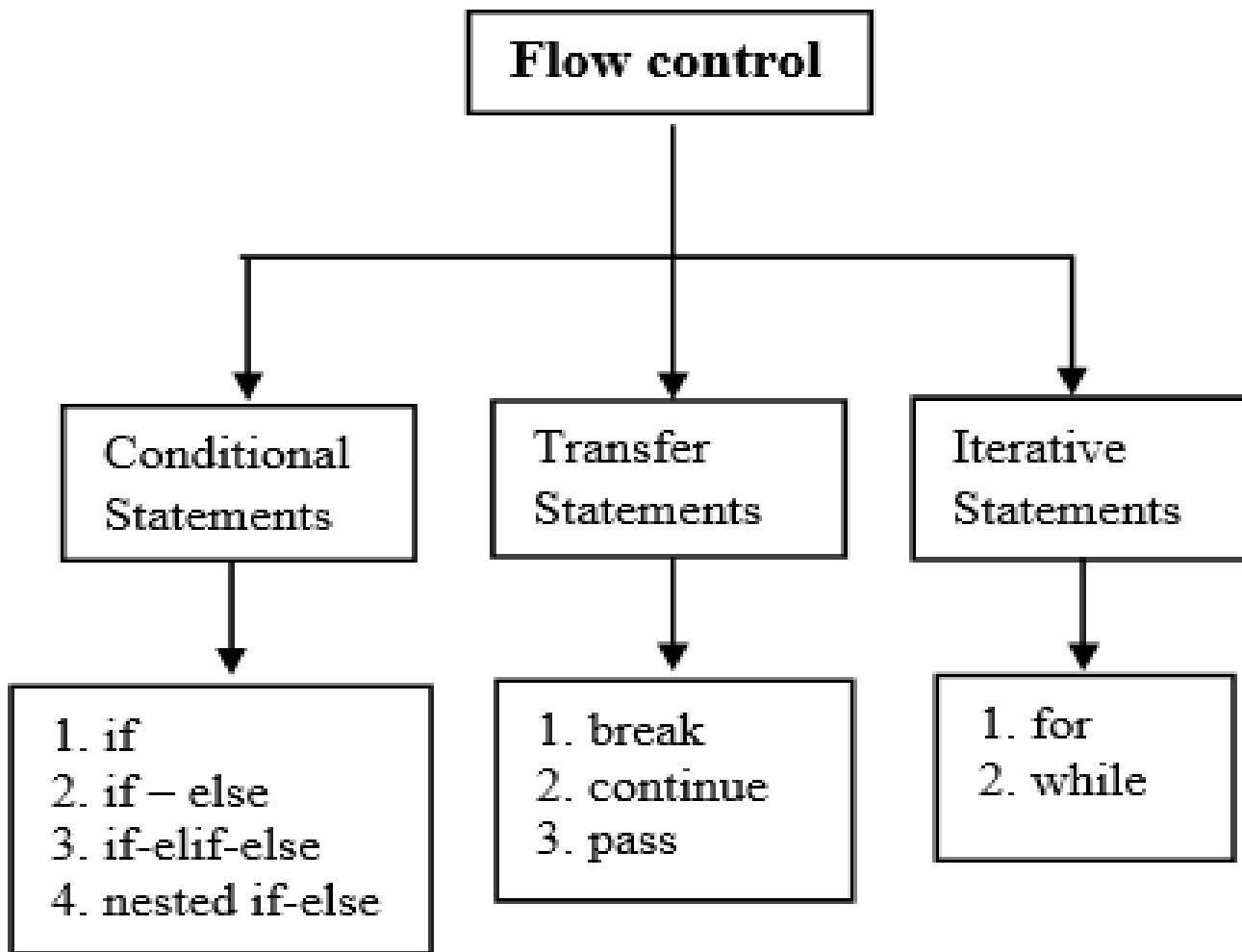
S = { 20, 'Jessa', 35.75}

- ✓ **Unordered**: Set doesn't maintain the order of the data insertion.
- ✓ **Unchangeable**: Set are immutable and we can't modify items.
- ✓ **Heterogeneous**: Set can contains data of all types
- ✓ **Unique**: Set doesn't allows duplicates items

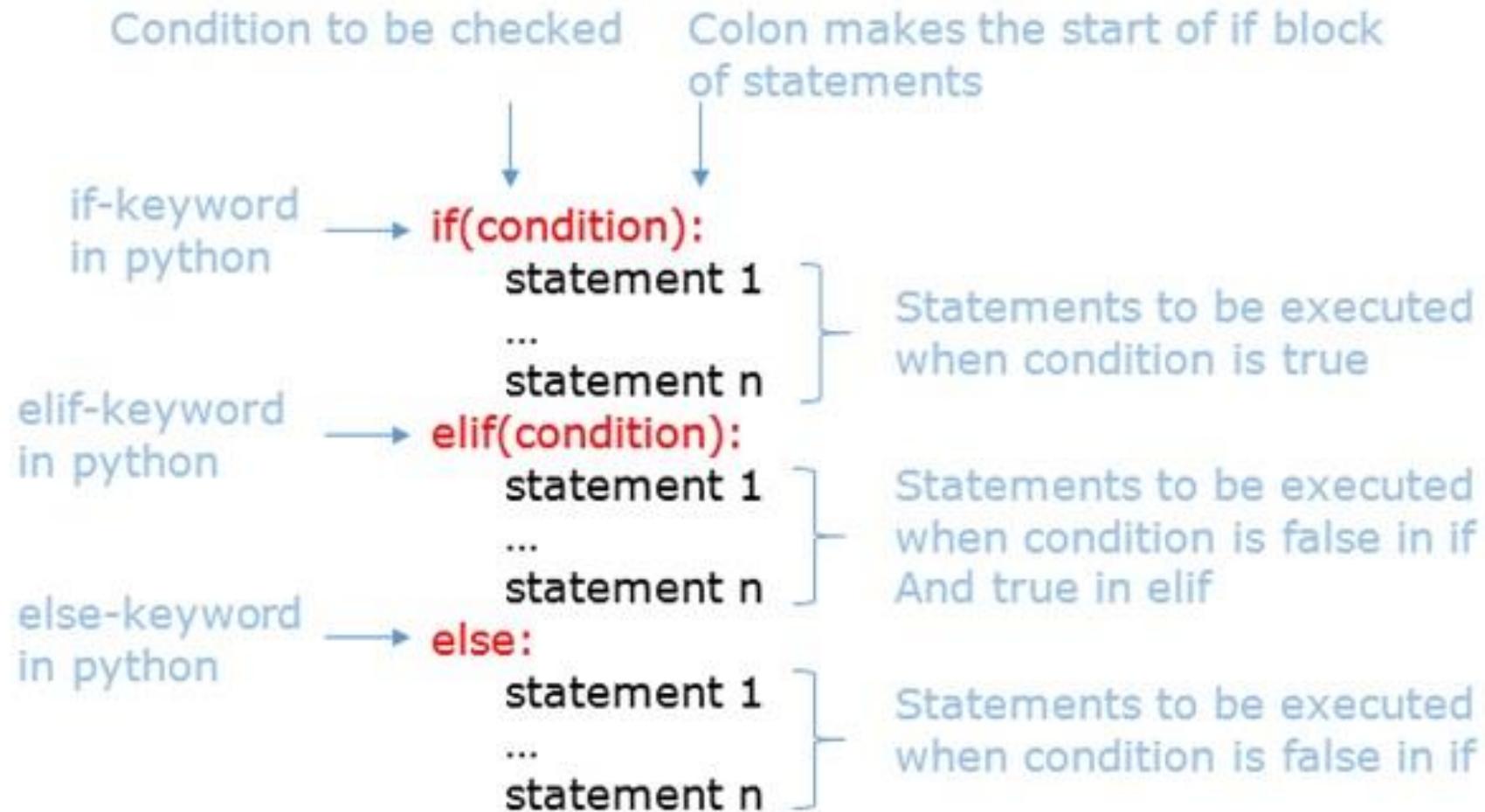
# Set Operations

Operation	Equivalent	Result
<code>len(s)</code>		number of elements in set <code>s</code> (cardinality)
<code>x in s</code>		test <code>x</code> for membership in <code>s</code>
<code>x not in s</code>		test <code>x</code> for non-membership in <code>s</code>
<code>s.issubset(t)</code>	<code>s &lt;= t</code>	test whether every element in <code>s</code> is in <code>t</code>
<code>s.issuperset(t)</code>	<code>s &gt;= t</code>	test whether every element in <code>t</code> is in <code>s</code>
<code>s.union(t)</code>	<code>s   t</code>	new set with elements from both <code>s</code> and <code>t</code>
<code>s.intersection(t)</code>	<code>s &amp; t</code>	new set with elements common to <code>s</code> and <code>t</code>
<code>s.difference(t)</code>	<code>s - t</code>	new set with elements in <code>s</code> but not in <code>t</code>
<code>s.symmetric_difference(t)</code>	<code>s ^ t</code>	new set with elements in either <code>s</code> or <code>t</code> but not both
<code>s.copy()</code>		new set with a shallow copy of <code>s</code>

# Flow Control



# if, elif, and else Statements



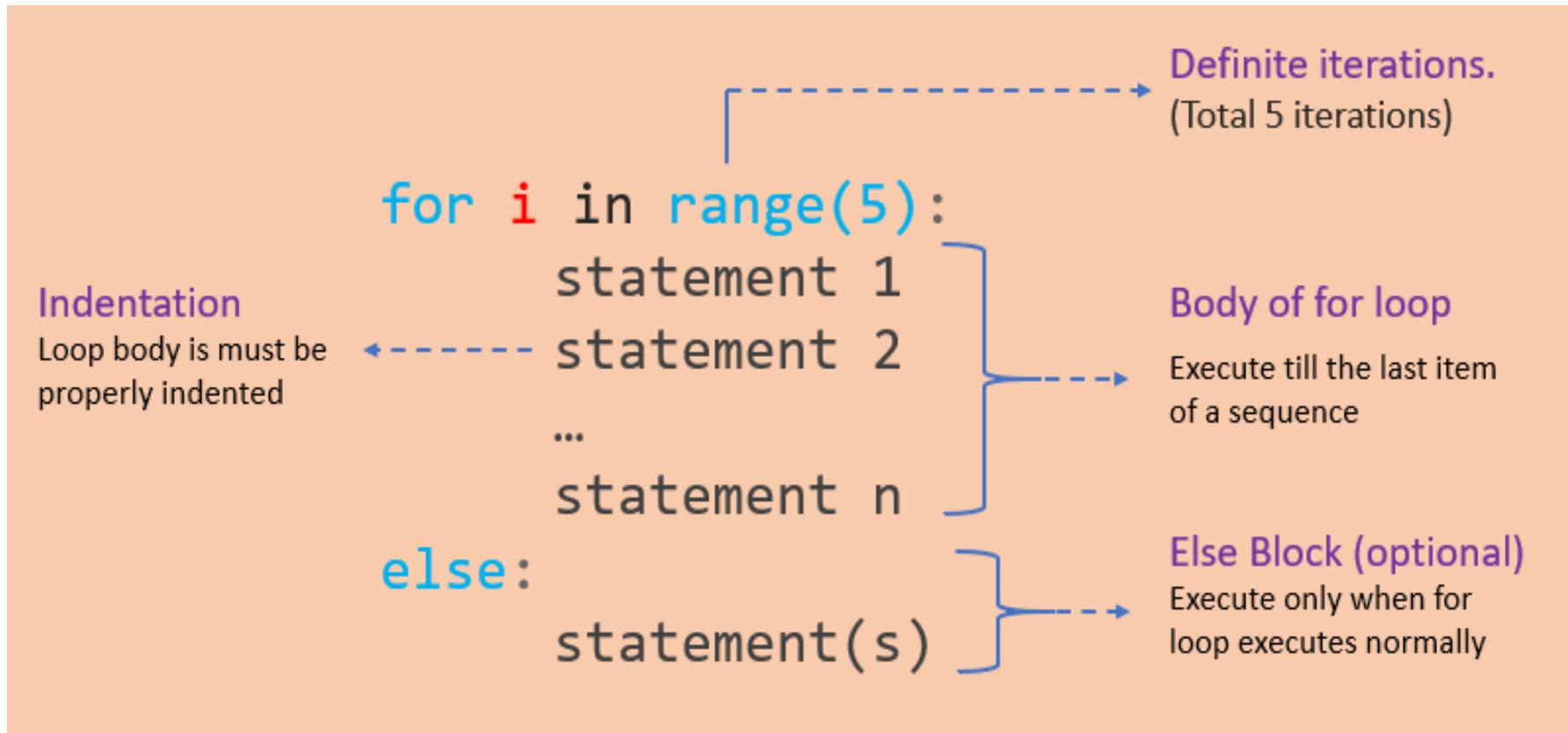
# Relational (comparison) operators

Operator	Description	Example
> (Greater than)	It returns True if the left operand is greater than the right	$x > y$ result is True
< (Less than)	It returns True if the left operand is less than the right	$x < y$ result is False
== (Equal to)	It returns True if both operands are equal	$x == y$ result is False
!= (Not equal to)	It returns True if both operands are equal	$x != y$ result is True
>= (Greater than or equal to)	It returns True if the left operand is greater than or equal to the right	$x >= y$ result is True
<= (Less than or equal to)	It returns True if the left operand is less than or equal to the right	$x <= y$ result is False

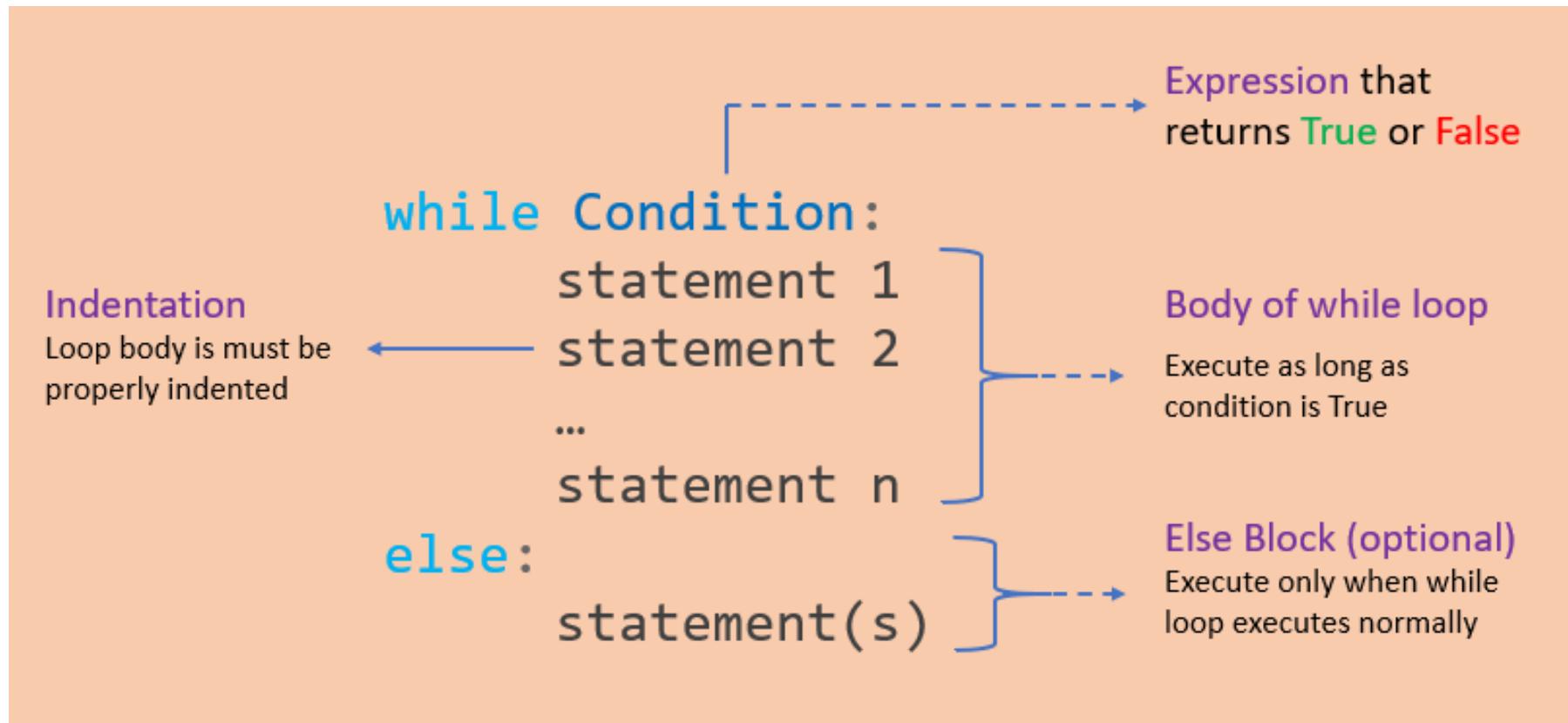
# Logical operators

Operator	Description	Example
and (Logical and)	True if both the operands are True	a and b
or (Logical or)	True if either of the operands is True	a or b
not (Logical not)	True if the operand is False	not a

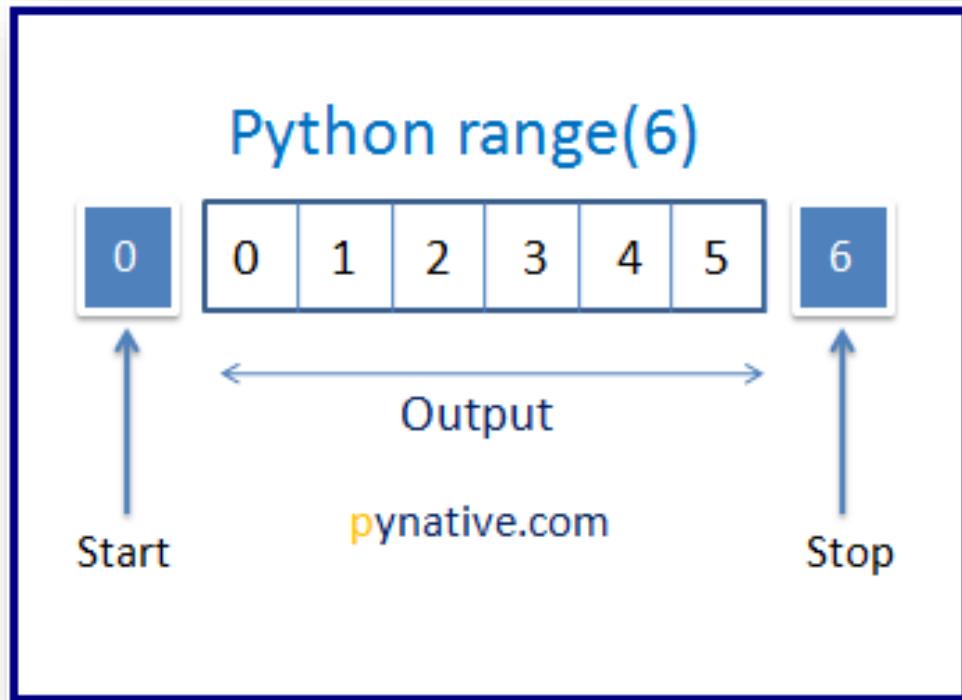
# For Loops



# While Loops



# Range Function



## How to use Python range(start, stop, step)

range() returns the immutable sequence of numbers starting from the given **start integer to the stop-step**. Each number is incremented by adding step value to its preceding number

range(0, 6, 1) → [0 | 1 | 2 | 3 | 4 | 5]

**Step.** (Optional) Specify the increment.  
Default is 1

**Stop.** (Required) specifying at which position to stop. Not part of the result

**Start.** (Optional) Start number of sequence. Default is 0

```
for i in range(6):  
    print(i)
```

It returns a range object not list  
type(range(6)) -> class 'range'

### Reverse range/ Decrementing

1 range(5, -1, -1) → [5 | 4 | 3 | 2 | 1 | 0]

2 reversed(range(6))

List from range x = list(range(6)) → [0 | 1 | 2 | 3 | 4 | 5]

range(-1, -11, -1)

Negative range from -1 to -10

range(start, stop+step, step)

Generate an inclusive range

range(0, 10)[5]

Access 5th number of a range()

range(10)[3:8]

Slice a range to from index 3 to 8

range(5).start, range(5).stop

Access range() attributes

# Range Function

```
>>> # One parameter          >>> # Two parameters        >>> # Three parameters      >>> # Going backwards
>>> for i in range(5):       >>> for i in range(3, 6):     >>> for i in range(4, 10, 2): >>> for i in range(0, -10, -2):
...     print(i)              ...     print(i)                 ...     print(i)                  ...
...                               ...                               ...
...                               3                               4
0                               4                               6
1                               5                               8
2
3
4
```

# Break, Continue, and Pass

Statement	Description
<b>break</b>	Terminate the <b>current loop</b> . Use the break statement to come out of the loop instantly.
<b>continue</b>	Skip the <b>current iteration</b> of a loop and move to the next iteration
<b>pass</b>	Do <b>nothing</b> . Ignore the condition in which it occurred and proceed to run the program as usual

# break

```
for i in sequence:  
    statement 1  
    statement 2  
    ...  
    if condition:  
        break  
        statement x  
        statement n
```

#Next statement after loop

Break Statement

Terminate loop  
immediately

Remaining  
body of loop

Body of a loop

Execute as long as  
condition is True

# continue

```
for i in sequence:  
    statement 1  
    statement 2  
    ...  
    if condition:  
        continue  
    statement x  
    statement n
```

continue  
Move to the  
next iteration

Statements  
Skipped

Body of a loop

Execute as long as  
condition is True

# Nested Loop

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

The diagram illustrates the structure of a nested loop. It features three curly braces: a green brace on the left labeled 'Outer Loop' enclosing the outermost 'for' loop; a red brace in the middle labeled 'Inner loop' enclosing the inner 'for' loop; and a purple brace on the right labeled 'Body of Outer loop' enclosing both the inner loop structure and the final 'print('')' statement. An arrow points from the text 'Body of inner loop' to the 'print(i\*j, end=" ")' line.

Python Program to Find the Factorial of a Number

Python program to print the duplicate elements of an array

Python program to find  $\sum_1^n n!$

Python program to print the elements of an array in reverse order

Python program to print the largest element in an array

Python program to sort the elements of an array in ascending order

Python Program to Remove Punctuation from a String

Python program to find the frequency of each element in the array



# **Reading and Writing to text files in Python**

# Opening a File

```
File_object = open(r"File_Name","Access_Mode")
```

```
# Open function to open the file "MyFile1.txt"  
# (same directory) in append mode and  
file1 = open("MyFile.txt","a")  
  
# store its reference in the variable file1  
# and "MyFile2.txt" in D:\Text in file2  
file2 = open(r"D:\Text\MyFile2.txt","w+")
```

# File Access Modes

1. **Read Only ('r')** : Open text file for **reading**. The handle is positioned at the **beginning** of the file. If the file does **not exists**, raises I/O **error**. This is also the default mode in which file is opened.
2. **Read and Write ('r+')** : Open the file for **reading** and **writing**. The handle is positioned at the **beginning** of the file. Raises I/O **error** if the file does **not exists**.
3. **Write Only ('w')** : Open the file for **writing**. For existing file, the data is truncated and **over-written**. The handle is positioned at the **beginning** of the file. **Creates** the file if the file does not exists.
4. **Write and Read ('w+')** : Open the file for **reading** and **writing**. For existing file, data is truncated and **over-written**. The handle is positioned at the **beginning** of the file.
5. **Append Only ('a')** : Open the file for **writing**. The file is **created** if it does not exist. The handle is positioned at the **end** of the file. The data being written will be **inserted** at the end, after the existing data.
6. **Append and Read ('a+')** : Open the file for **reading** and **writing**. The file is **created** if it does not exist. The handle is positioned at the **end** of the file. The data being written will be **inserted** at the end, after the existing data.

# Closing a file

```
# Opening and Closing a file "MyFile.txt"  
# for object name file1.  
file1 = open("MyFile.txt","a")  
file1.close()
```

# Reading from a file

There are three ways to read data from a text file.

**1.read()** : Returns the read bytes in form of a string. Reads n bytes, if no n specified, reads the **entire file**.

**File\_object.read([n])**

**2.readline()** : Reads a line of the file and returns in form of a string. For specified n, reads at most n bytes.

However, does not reads more than **one line**, even if n exceeds the length of the line.

**File\_object.readline([n])**

**3.readlines()** : Reads **all the lines** and return them as each line a string element in a **list**.

**File\_object.readlines()**

# Writing to a file

There are two ways to write in a file.

1. **write()** : Inserts the string str1 in a single line in the text file.

**File\_object.write(str1)**

2. **writelines()** : For a list of string elements, each string is inserted in the text file. Used to insert multiple strings at a single time.

**File\_object.writelines(L) for L = [str1, str2, str3]**

# **Using write along with the with() function**

# Example

```
# Python code to illustrate split() function  
with open("file.txt", "r") as file:  
    data = file.readlines()  
    for line in data:  
        word = line.split()  
        print (word)
```

# Copy contents of one file to another file

```
# open both files
```

```
with open('first.txt','r') as firstfile, open('second.txt','a') as secondfile:
```

```
    # read content from first file
```

```
    for line in firstfile:
```

```
        # append content to second file
```

```
        secondfile.write(line)
```

# Copy all the content of one file to another file in uppercase

```
# To open the first file in read mode  
f1 = open("sample file 1.txt", "r")  
  
# To open the second file in write mode  
f2 = open("sample file 2.txt", "w")  
  
# For loop to traverse through the file  
for line in f1:  
  
    # Writing the content of the first  
    # file to the second file  
  
    # Using upper() function to  
    # capitalize the letters  
    f2.write(line.upper())
```

# Copy odd lines of one file to other

```
# open file in read mode
fn = open('bcd.txt', 'r')
# open other file in write mode
fn1 = open('nfile.txt', 'w')
# read the content of the file line by line
cont = fn.readlines()
type(cont)
for i in range(0, len(cont)):
    if(i % 2 != 0):
        fn1.write(cont[i])
    else:
        pass
```

# Count number of lines in a text file in Python

```
file = open("gfg.txt","r")
Counter = 0

# Reading from file
Content = file.read()
CoList = Content.split("\n")

for i in CoList:
    if i:
        Counter += 1
print("This is the number of lines in the file")
print(Counter)
```

# Exercise

- Get number of characters, words, spaces and lines in a file
- Read lines from this file and split line on last delimiter in Python and save it to another file

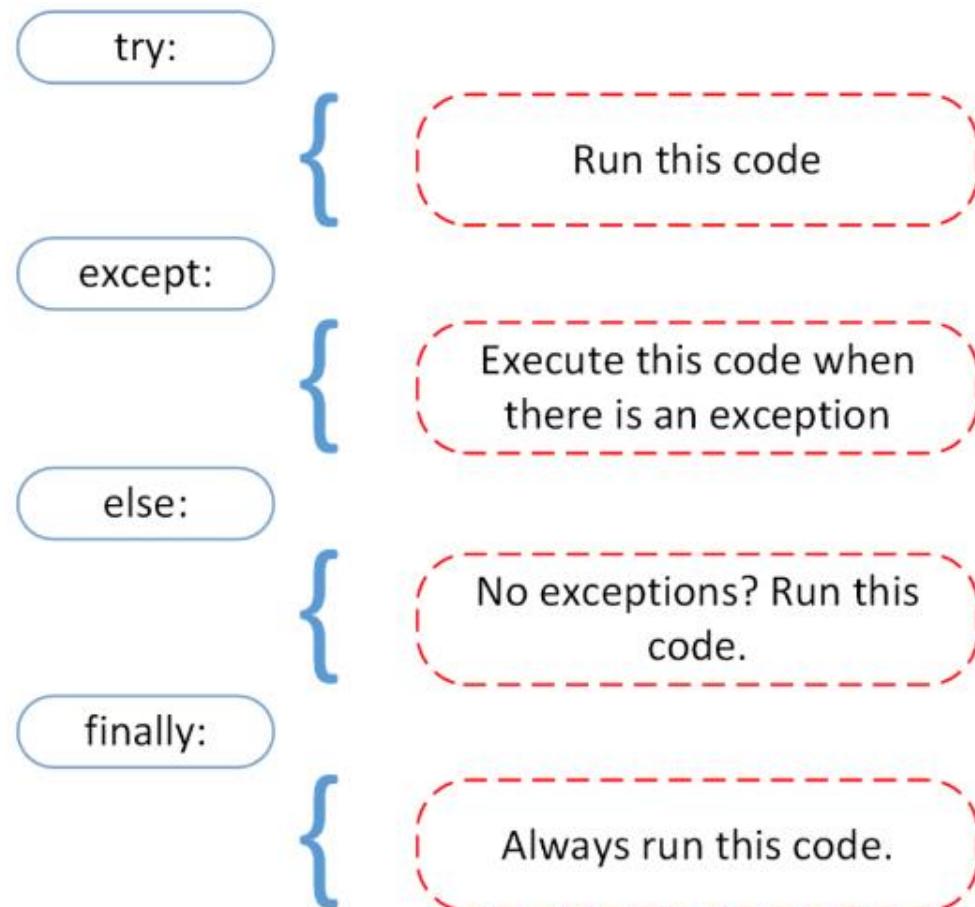
```
first_name1:last_name1:mail1@mail.com:address1
first_name2:last_name2:mail2@mail.com:address2
first_name3:last_name3:mail3@mail.com:address3
first_name4:last_name5:mail4@mail.com:address4
first_name5:last_name6:mail5@mail.com:address5
first_name6:last_name7:mail6@mail.com:address6
first_name7:last_name8:mail7@mail.com:address7
```



# Errors Types

1. Syntax errors
2. Logical errors (Exceptions)

# Errors and Exception Handling



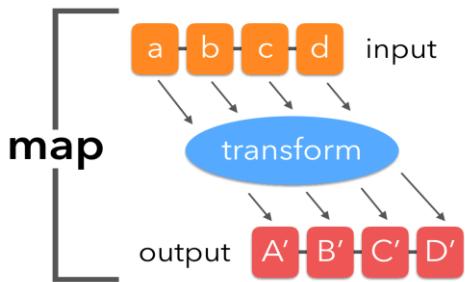
# try-except vs. raise

## try - except

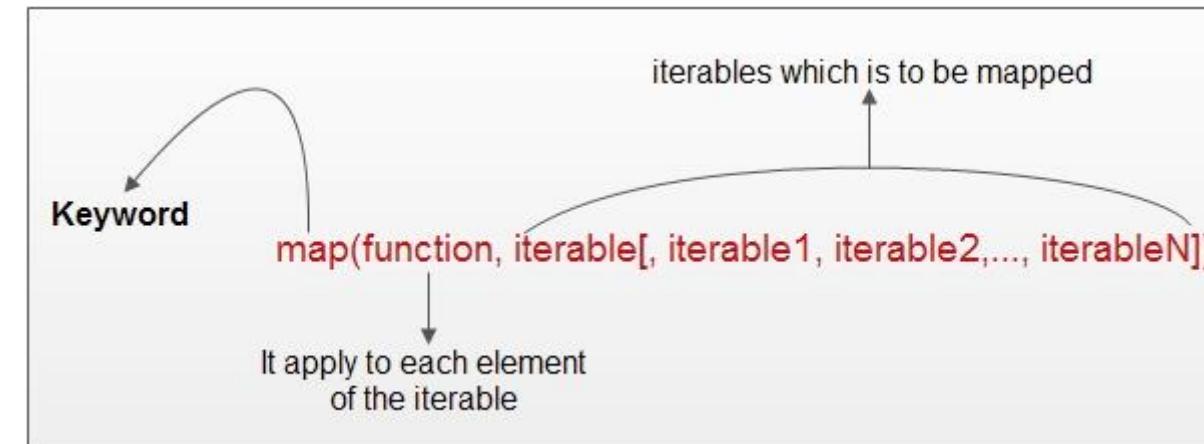
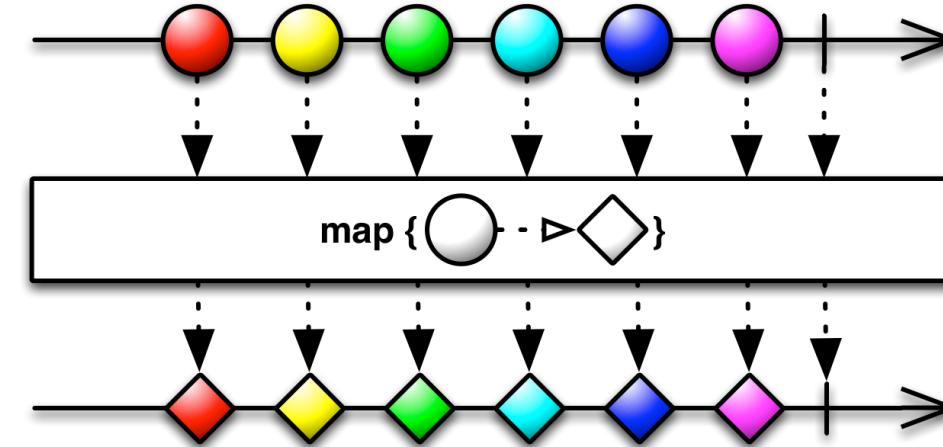
- Avoid errors being produced
- Still execute subsequent code

## raise

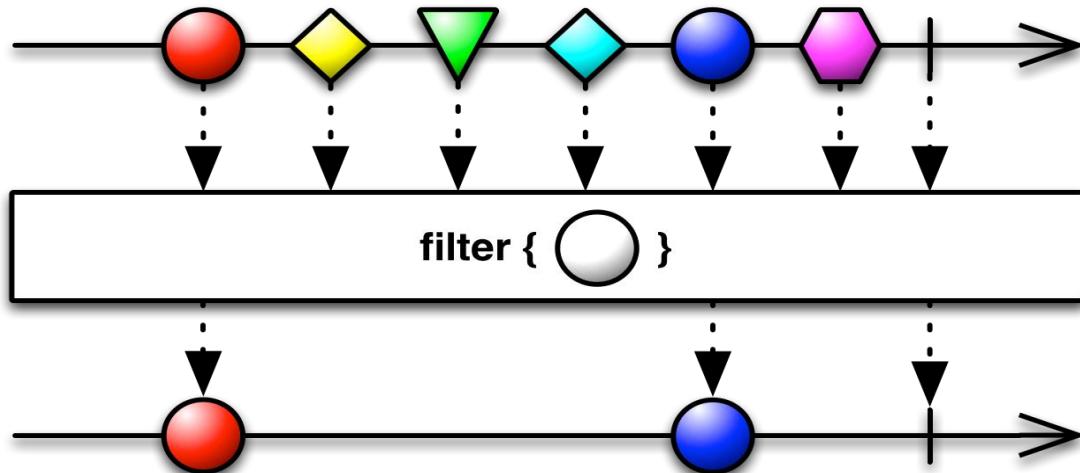
- Will produce an error
- Avoid executing subsequent code



# map



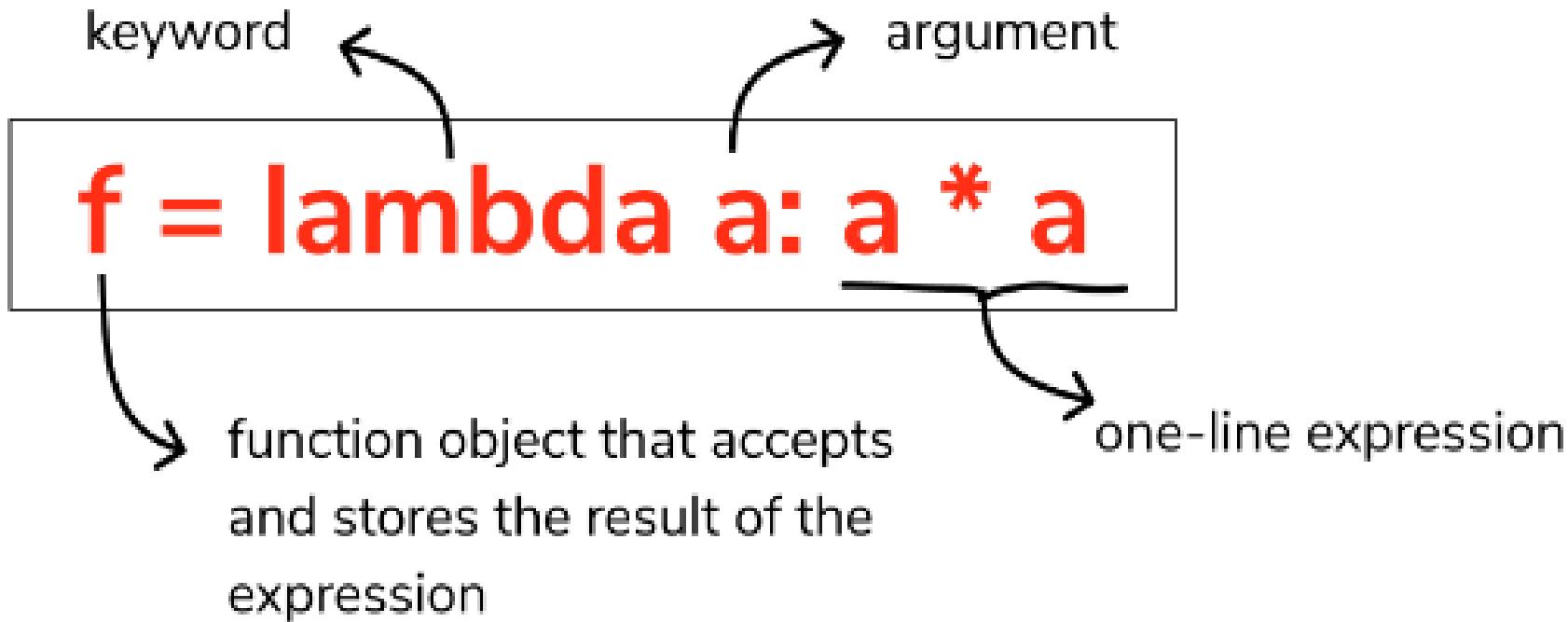
# filter



```
filter(function, iterable)
```

Returns a sequence from those elements of iterable for which the function returns true.

# Lambda



# Lambda functions with iterables

- `map()` applies a function to all elements in an iterable

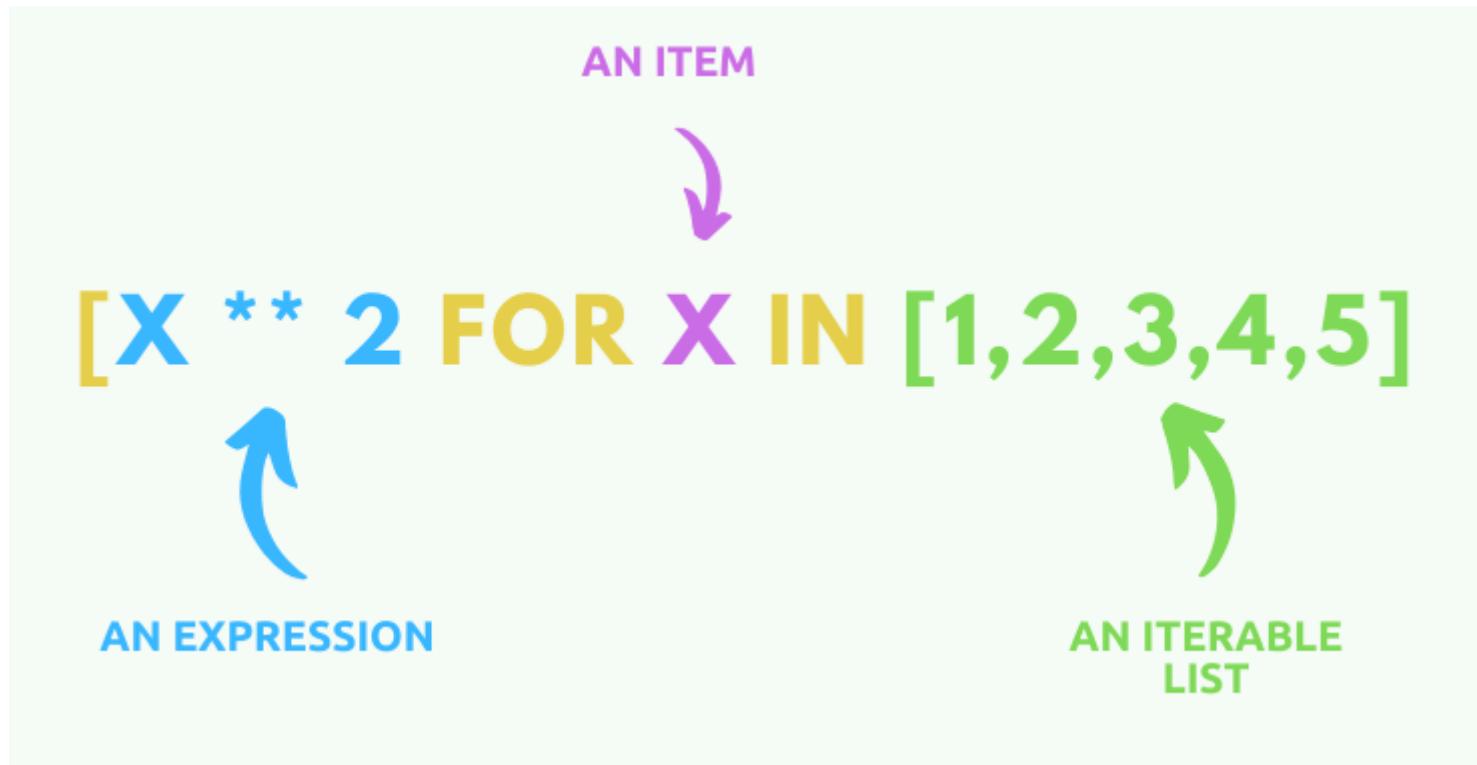
```
names = ["john", "sally", "leah"]
# Apply a lambda function inside map()
capitalize = map(lambda x: x.capitalize(), names)
print(capitalize)
```

To convert map object we have to put it as data structure  
<map object at 0x7fb200529c10>

```
# Convert to a list
list(capitalize)
```

```
['John', 'Sally', 'Leah']
```

# List comprehension



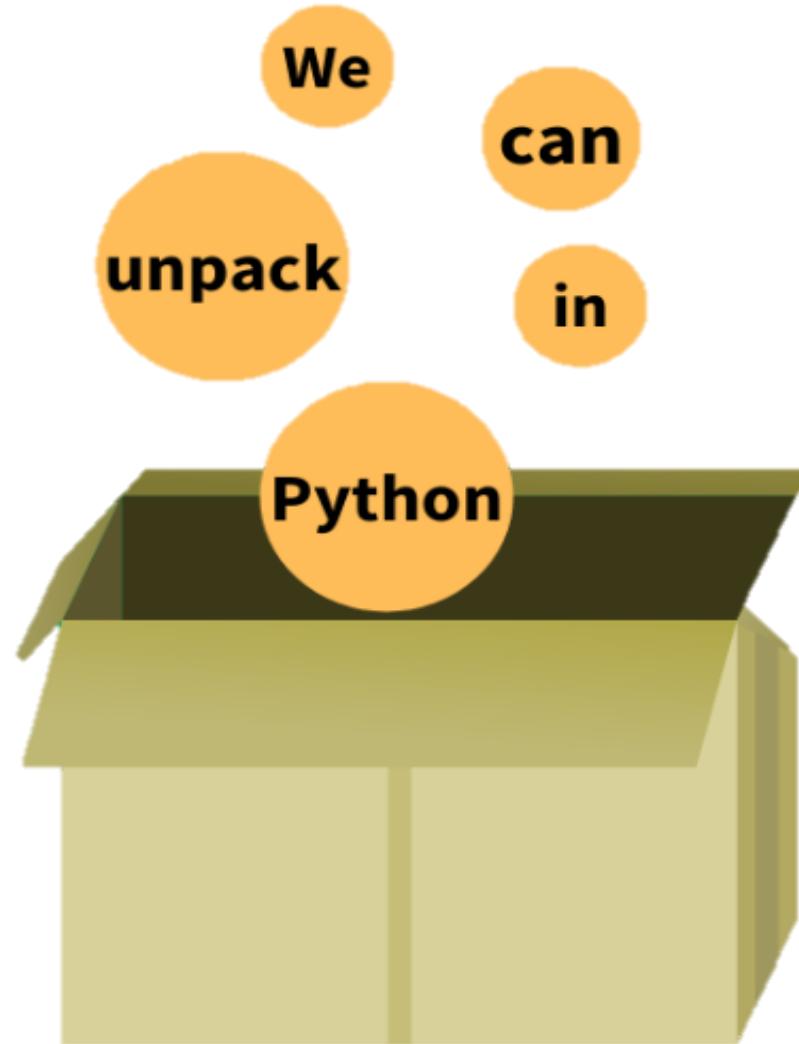
Do this

For this collection

In this situation

`[x**2 for x in range(0, 50) if x % 3 == 0]`

# Unpacking



# \*args and \*\*Kwargs

## Function definition

```
def fun_name(*args)      → Packing arguments  
    statement(s)
```

## Function call

```
fun_name(a,b,c,d)      → Actual arguments
```

**\*\*kwargs means**  
**passing an arbitrary**  
**number of arguments**  
**with keywords**



Parameter 1

Parameter 2

Parameter 3

.....

**\*args means passing an arbitrary number of arguments**

## Function definition

```
def fun_name(**kwargs) → Packing keyword arguments  
    statement(s)
```

## Function call

```
fun_name(a=1,b=2,c=3) → keyword arguments
```

Using map() function and lambda and count() function create a list which consists of the number of occurrence of letter: a.

Using map() function and lambda and count() function create a list consisted of the number of occurrence of both letters: A and a.

Using filter() function filter the list so that only negative numbers are left.

Using map() and filter() functions add 2000 to the values below 8000.

Write a program that reads the sentence and gives the output as the length of each word in a sentence in the form of a list.



Using map() function and lambda and count() function create a list which consists of the number of occurrence of letter: a.

```
lst2 = list(map(lambda x: x.count("a"), lst1))
```

Using map() function and lambda and count() function create a list consisted of the number of occurrence of both letters: A and a.

```
lst2 = list(map(lambda x: x.lower().count("a"), lst1))
```

Using filter() function filter the list so that only negative numbers are left.

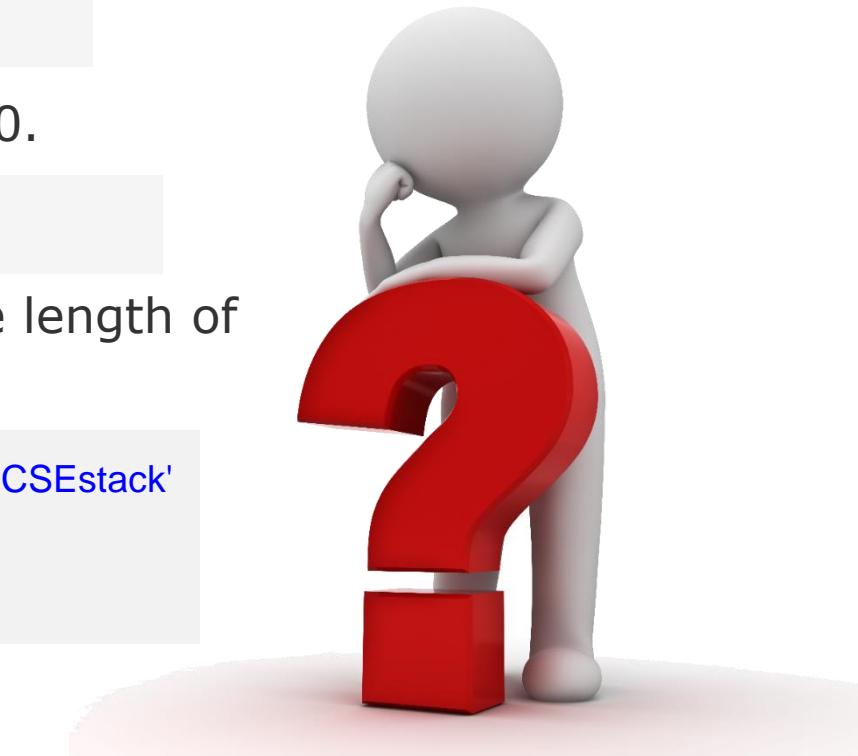
```
lst2 = list(filter(lambda x: x<0, lst1))
print(lst2)
```

Using map() and filter() functions add 2000 to the values below 8000.

```
lst2 = list(map(lambda x: x+2000, filter(lambda x: x<8000, lst1)))
```

Write a program that reads the sentence and gives the output as the length of each word in a sentence in the form of a list.

```
sentence = 'I am learning Python programming with CSEstack'
words = sentence.split()
lengths = map(lambda word: len(word), words)
```



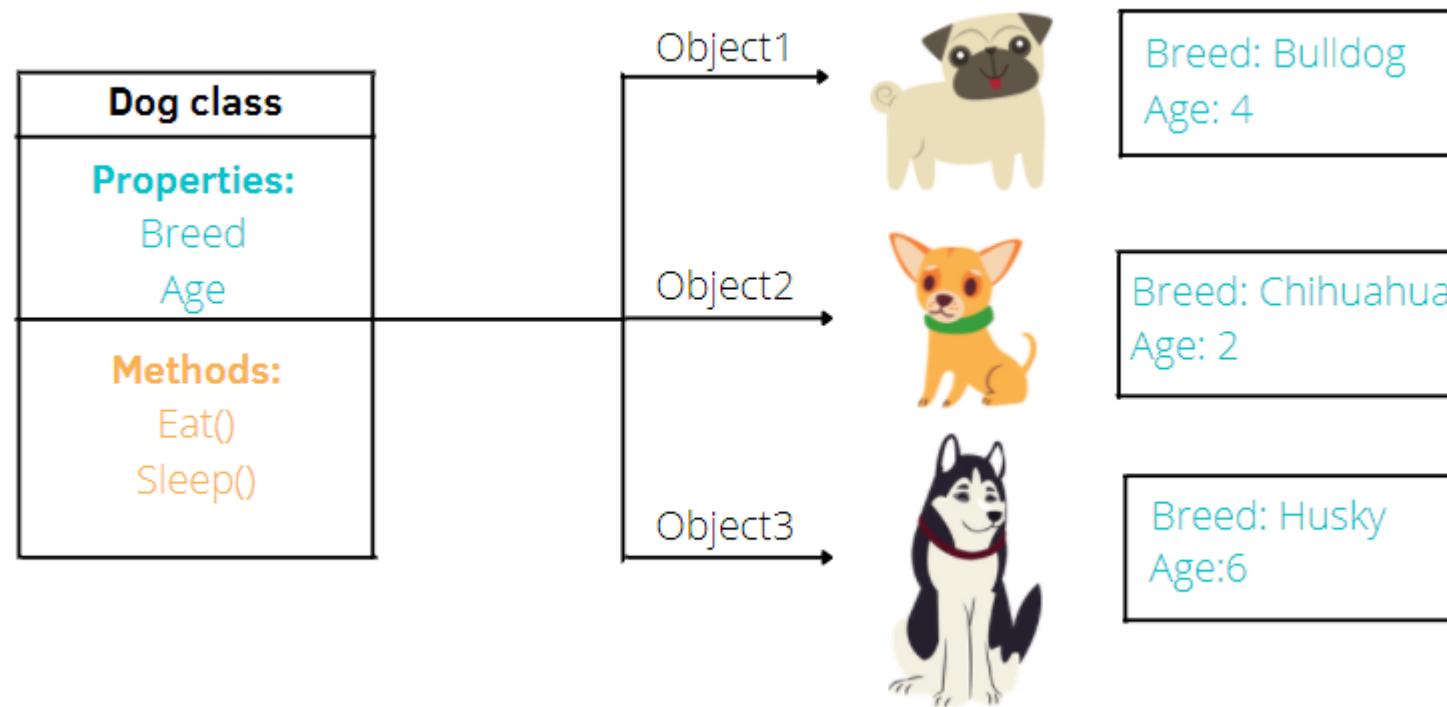
# Functions

```
Function Name    Parameters  
↑             ↗      ↑  
def add(num1, num2):  
    print("Number 1:", num1)  
    print("Number 2:", num1)  
    addition = num1 + num2  
  
return addition → Return Value
```

Function Body

```
res = add(2, 4) → Function call  
print(res)
```

# Classes



# Classes

The class is declared  
with the keyword  
`class`

```
class Rational:  
    def __init__(self, numerator, denominator):  
        self.numerator = numerator  
        self.denominator = denominator
```

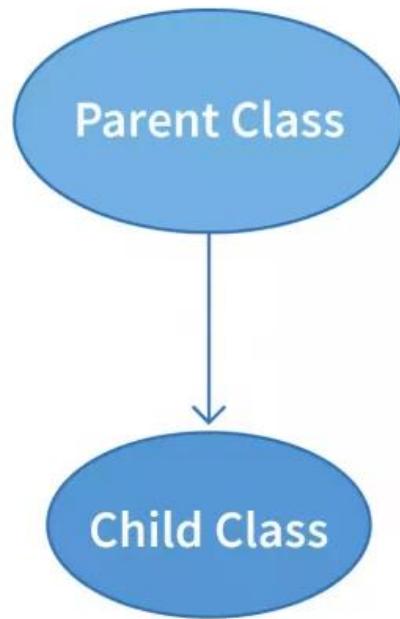
Functions and variables  
declared inside the class  
block should have  
indentation.

Class instantiation automatically  
invokes an `__init__()`  
method if it exists.

# Classes

```
class Books:  
    def __init__(self, title, cost):  
        self.title = title  
        self.cost = cost  
    def gettitle(self):  
        print("Title of the book is: "+self.title)  
  
python_book = Books("Learn and Practice Python Programming", 136)  
print(python_book.title)  
print(python_book.cost)  
python_book.gettitle()
```

# Inheritance



Child  
Inherits  
qualities  
from parent

# Inheritance

```
class ParentClass:

    def feature_1(self):
        print('feature_1 from ParentClass is running...')

    def feature_2(self):
        print('feature_2 from ParentClass is running...')


class ChildClass(ParentClass):

    def feature_3(self):
        print('feature_3 from ChildClass is running...')

obj = ChildClass()
obj.feature_1()
obj.feature_2()
obj.feature_3()
```

Define a class called Songs, it will show the lyrics of a song. Its `__init__()` method should have two arguments:`self` and `lyrics`. `lyrics` is a list. Inside your class create a method called `sing_me_a_song` that prints each element of `lyrics` on his own line. Define a variable:

```
happy_bday = Song(["May god bless you, ", "Have a sunshine on you,", "Happy Birthday to you !"])
```

Call the `sing_me_song` method on this variable.

```
>>> class Song(object):
... def __init__(self, lyrics):
...     self.lyrics=lyrics
... def sing_me_a_song(self):
...     for line in self.lyrics:
...         print line
```

```
happy_bday = Song(["May god bless you, ", ... "Have a sunshine on you,", ... "Happy Birthday to you !"])
print happy_bday.sing_me_a_song()
```



# Python Modules

```
def add(a, b):  
    return a + b  
  
def sub(a, b):  
    return a - b  
  
def mul(a, b):  
    return a * b  
  
def div(a, b):  
    return a / b
```

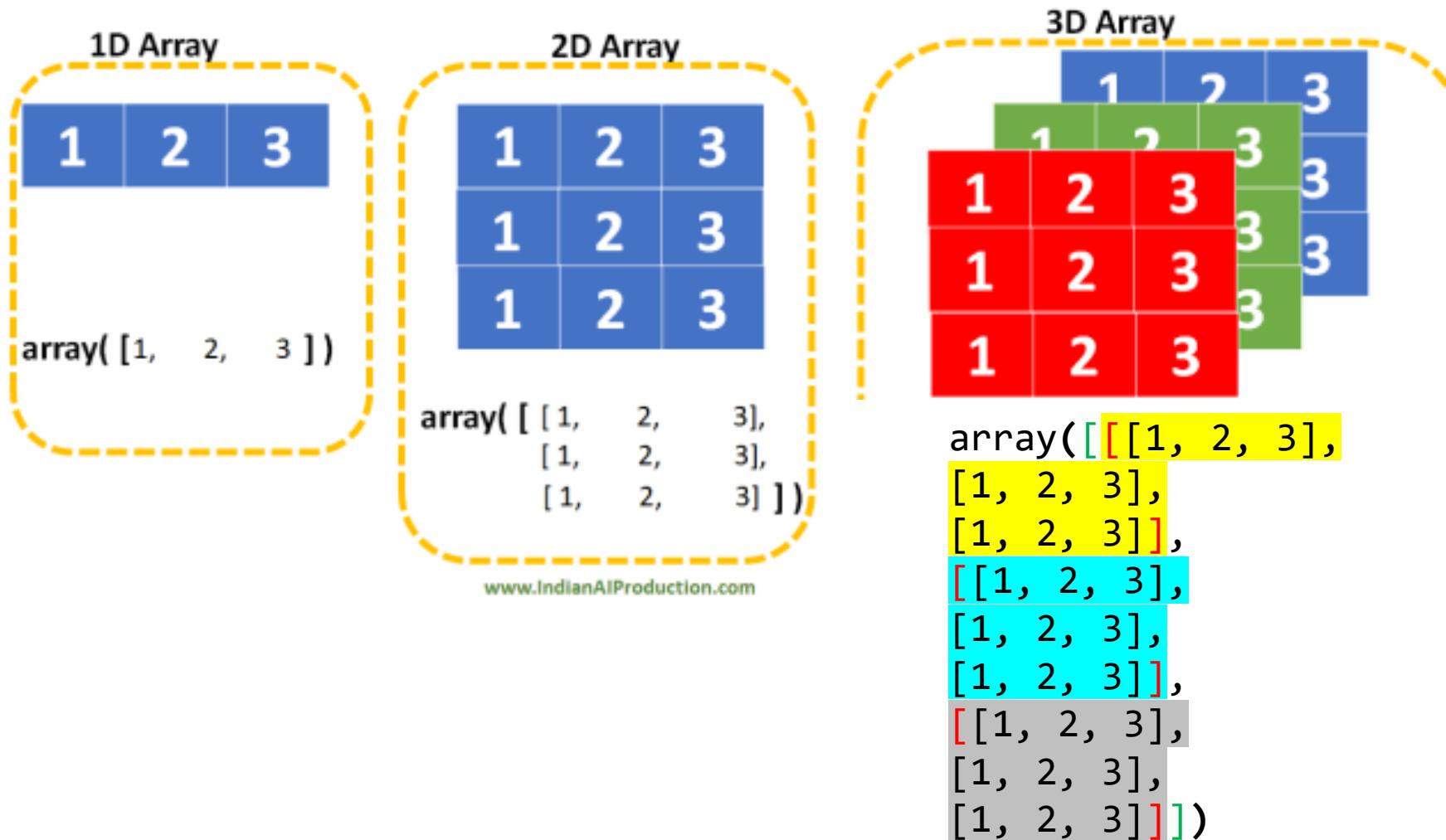
```
import Calci  
  
print(Calci.add(10, 5))  
print(Calci.sub(10, 5))  
print(Calci.mul(10, 5))  
print(Calci.div(10, 5))
```

# NumPy Basics

# NumPy Agenda

- NumPy Intro
- Creating Arrays
- NumPy Array Indexing
- NumPy Array Slicing
- NumPy Data Types
- NumPy Copy vs View
- NumPy Array Shape
- NumPy Array Reshape
- NumPy Array Filter

# Creating NumPy Arrays



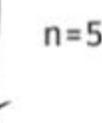
# Creating NumPy Array

`np.array([1,2,3])`



1
2
3

`np.arange(1, n+1)`



$n=5$

1	2	3	4	5
---	---	---	---	---

`np.ones(3)`



1
1
1

`np.zeros(3)`



0
0
0

`np.random.random(3)`



0.5967
0.0606
0.2223

`np.ones((3,2))`

1	1
1	1
1	1

3

`np.zeros((3,2))`

0	0
0	0
0	0

`np.random.random((3,2))`

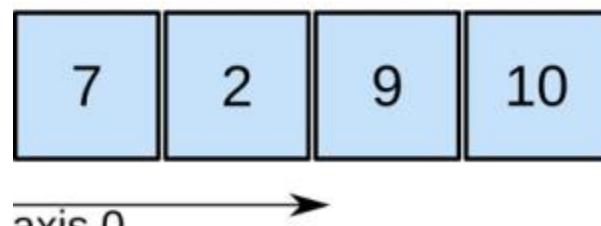


0.37	0.88
0.75	0.79
0.63	0.16

# NumPy Arrays Shape

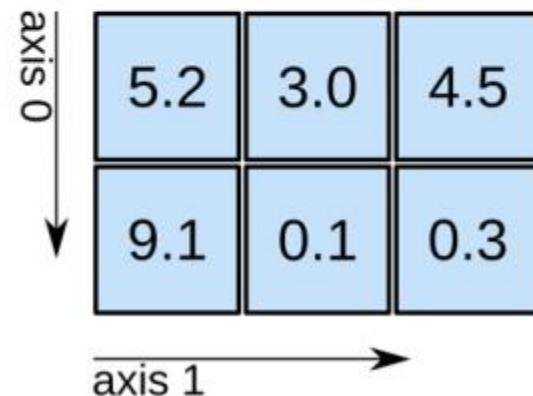
## 3D array

## 1D array

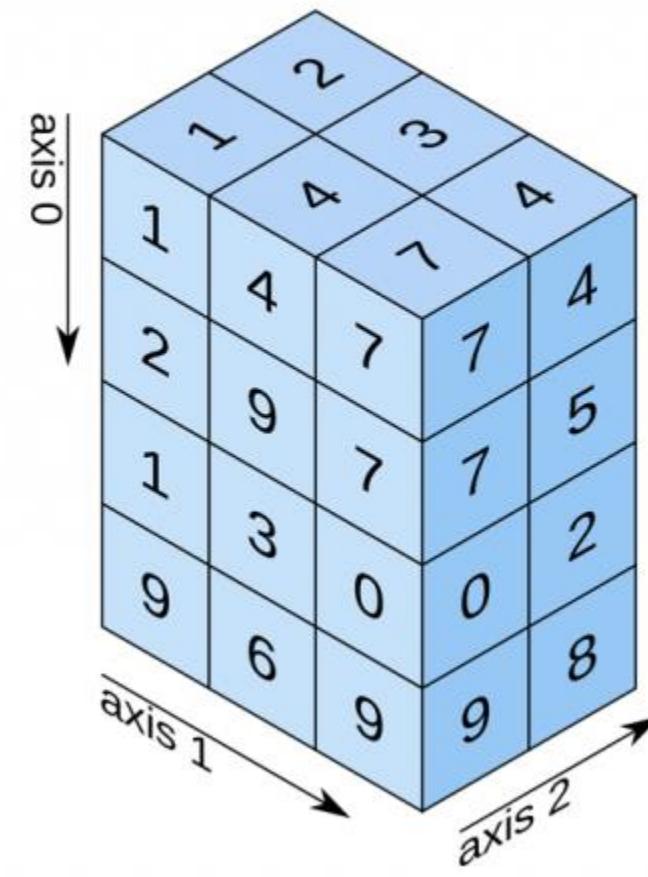


shape: (4,)

## 2D array

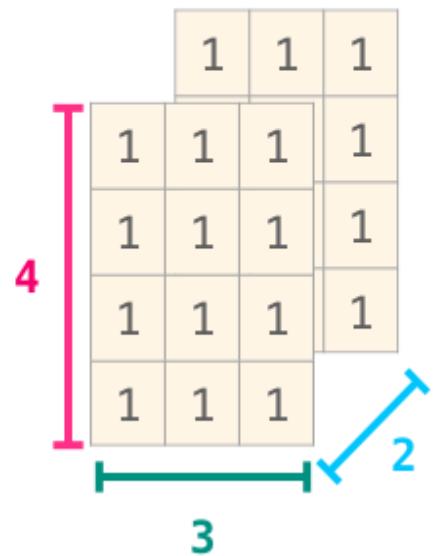


shape: (2, 3)

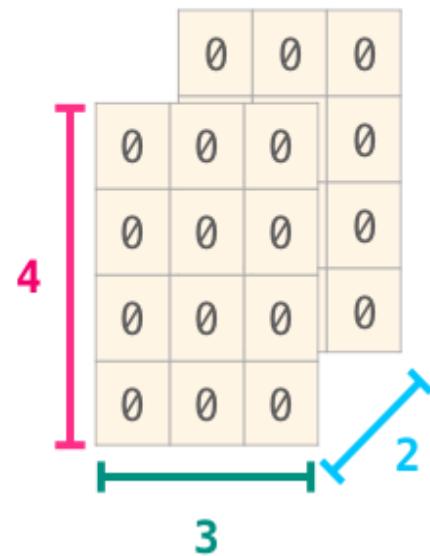


# NumPy Arrays Shape

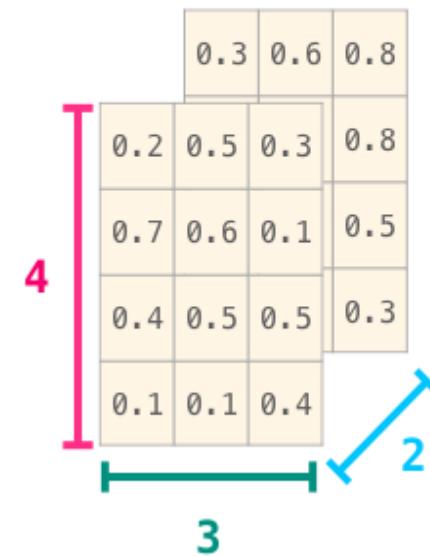
`np.ones((4,3,2))`



`np.zeros((4,3,2))`

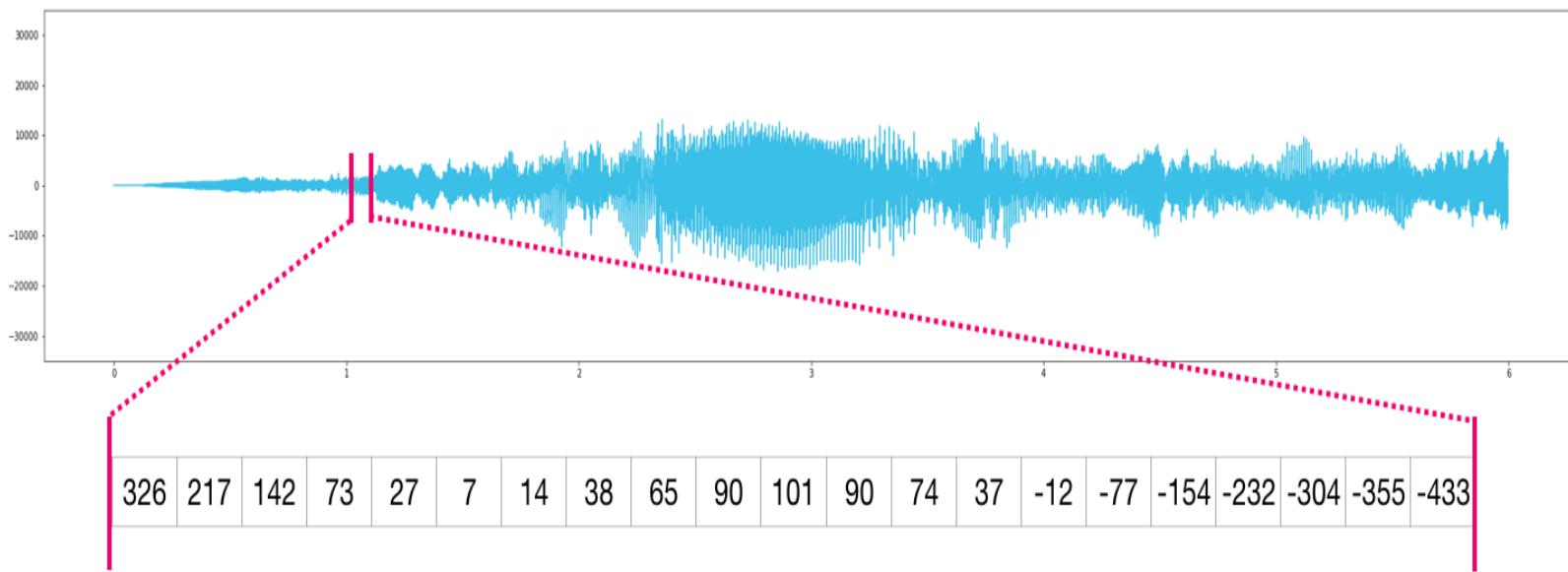


`np.random.random((4,3,2))`



# Audio and Timeseries

- An audio file is a one-dimensional array of samples. Each sample is a number representing a tiny chunk of the audio signal.
- CD-quality audio may have 44,100 samples per second and each sample is an integer between -32767 and 32768.
- Meaning if you have a ten-seconds WAVE file of CD-quality, you can load it in a NumPy array with length  $10 * 44,100 = 441,000$  samples.
- Want to extract the first second of audio? simply load the file into a NumPy array that we'll call `audio`, and get `audio[:44100]`.

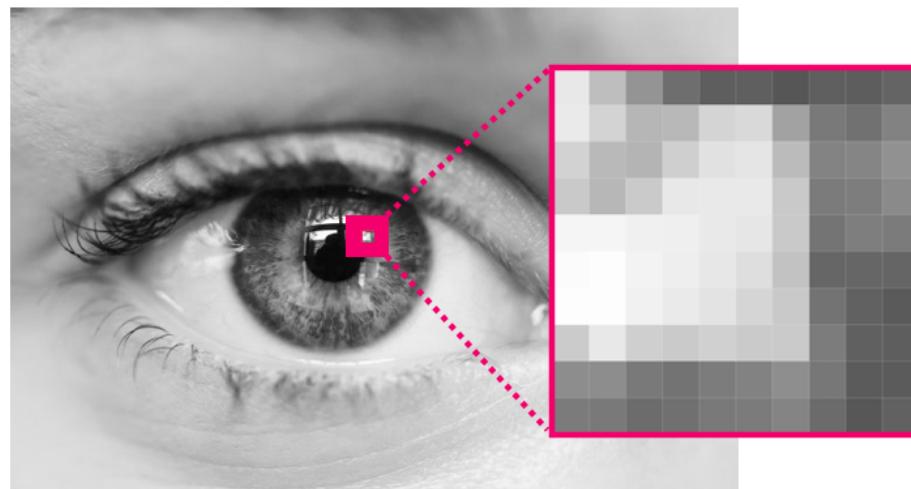


# Image

- An image is a matrix of pixels of size (height x width).

- If the image is black and white (a.k.a. grayscale), each pixel can be represented by a single number (commonly between 0 (black) and 255 (white)).

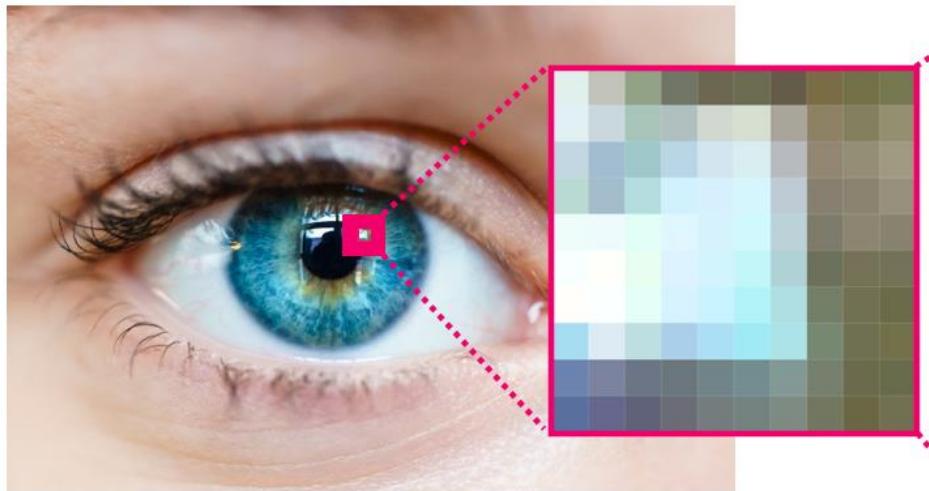
Want to crop the top left  $10 \times 10$  pixel part of the image? Just tell NumPy to get you `image[:10,:10]`.



230	194	147	108	90	98	84	96	91	101
237	206	188	195	207	213	163	123	116	128
210	183	180	205	224	234	188	122	134	147
198	189	201	227	229	232	200	125	127	135
249	241	237	244	232	226	202	116	125	126
251	254	241	239	230	217	196	102	103	99
243	255	240	231	227	214	203	116	95	91
204	231	208	200	207	201	200	121	95	95
144	140	120	115	125	127	143	118	92	91
121	121	108	109	122	121	134	106	86	97

# Image

- If the image is colored, then each pixel is represented by three numbers - a value for each of red, green, and blue. In that case we need a 3rd dimension (because each cell can only contain one number).
- So a colored image is represented by an ndarray of dimensions: (height x width x 3).



233	188	137	96	90	95	63	73	73	82	
237	202	159	120	105	110	88	107	112	121	109
226	191	147	110	101	112	98	123	110	119	142
221	191	176	182	203	214	169	144	133	145	155
185	160	161	184	205	223	186	137	147	161	140
181	174	189	207	206	215	194	136	142	151	133
246	237	237	231	208	206	192	122	143	144	111
254	254	241	224	199	192	181	99	122	117	107
239	248	232	207	187	182	184	110	114	110	113
193	215	193	167	158	164	181	114	112	111	105
113	119	110	111	113	123	135	120	108	106	113
93	97	91	103	107	111	122	112	104	114	

# Language

- If we're dealing with text, the story is a little different. The numeric representation of text requires a step of building a vocabulary (an inventory of all the unique words the model knows) and an [embedding step](#). Let us see the steps of numerically representing this (translated) quote by an ancient spirit:
- “Have the bards who preceded me left any theme unsung?”
- A model needs to look at a large amount of text before it can numerically represent the anxious words of this warrior poet. We can proceed to have it process a [small dataset](#) and use it to build a vocabulary (of 71,290 words):

Model Vocabulary

#	
0	the
1	of
2	and
...	...
71,289	dolophine

The sentence can then be broken into an array of tokens (words or parts of words based on common rules):

have	the	bards	who	preceded	me	left	any	theme	unsung
------	-----	-------	-----	----------	----	------	-----	-------	--------

We then replace each word by its id in the vocabulary table:

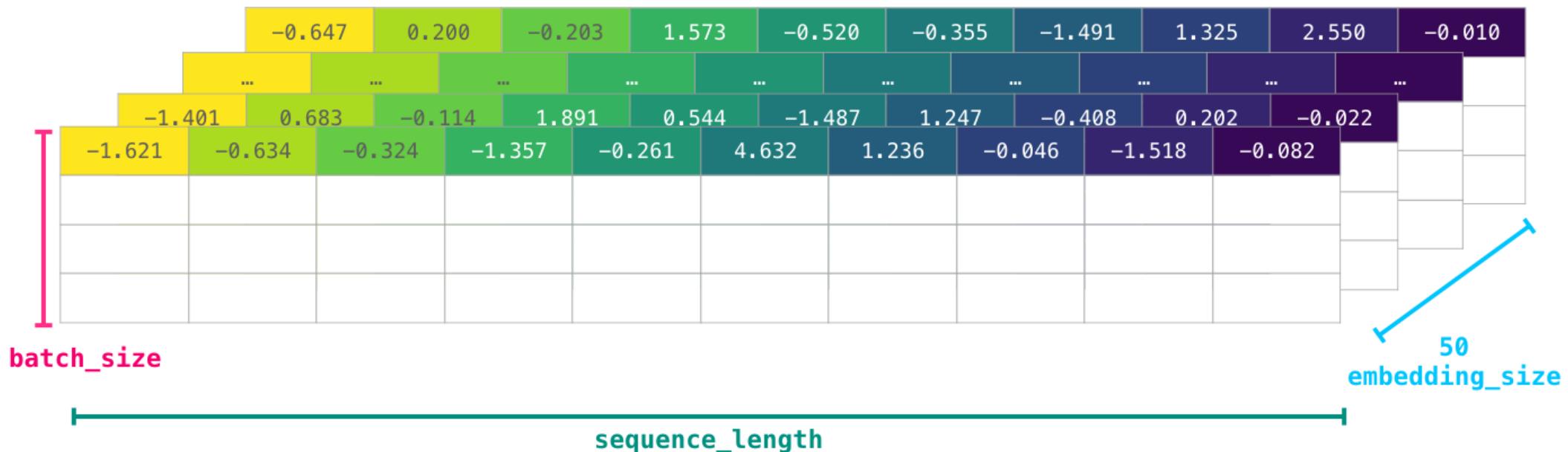
38	0	29104	56	7027	745	225	104	2211	66609
----	---	-------	----	------	-----	-----	-----	------	-------

These ids still don't provide much information value to a model. So before feeding a sequence of words to a model, the tokens/words need to be replaced with their embeddings (50 dimension [word2vec embedding](#) in this case):

	have	the	bards	who	preceded	me	left	any	theme	unsung
0	-1.621	-0.634	-0.324	-1.357	-0.261	4.632	1.236	-0.046	-1.518	-0.082
1	-1.401	0.683	-0.114	1.891	0.544	-1.487	1.247	-0.408	0.202	-0.022
	...	...	...	...	...	...	...	...	...	...
49	-0.647	0.200	-0.203	1.573	-0.520	-0.355	-1.491	1.325	2.550	-0.010

# Language

- You can see that this NumPy array has the dimensions [embedding\_dimension x sequence\_length]. In practice these would be the other way around, but I'm presenting it this way for visual consistency.
- For performance reasons, deep learning models tend to preserve the first dimension for batch size (because the model can be trained faster if multiple examples are trained in parallel).
- This is a clear case where `reshape()` becomes super useful. A model like [BERT](#), for example, would expect its inputs in the shape: [batch\_size, sequence\_length, embedding\_size].



# NumPy Array Indexing

**arr[      ,      ]**

rows selector                          columns selector

  :                                        :

  [1, 2, 3]                            [1, 2, 3]

  0:3                                    0:3

  0                                     0

  -1                                    -1

# NumPy Array Indexing

	data
0	1
1	2
2	3

	data[0]
	1

	data[1]
	2

	data[0:2]
	1
	2

	data[1:]
	2
	3

	data[-2:]
	2
	3

	data	
0	1	-2
1	2	-1
2	3	
3		

# NumPy Array Indexing

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



1	2
3	4
5	6

	0	1
0	1	2
1	3	4
2	5	6

	0	1
0	1	2
1	3	4
2	5	6

	0	1
0	1	2
1	3	4
2	5	6

	0	1
0	1	2
1	3	4
2	5	6

# Basic array operations



`data + ones` = **data** + **ones** =

**data**      **ones**

1	1
2	1

=

2
3

**data** - **ones** =

**data** \* **data** =

**data** / **data** =

1	1
2	1

-

1
1

=

0
1

\*

1	1
2	2

=

1
4

/

1	1
2	2

=

1
1

# Broadcasting

$$\begin{array}{c|c} 1 \\ \hline 2 \end{array} * \mathbf{1.6} = \begin{array}{c|c} 1 \\ \hline 2 \end{array} * \begin{array}{c|c} 1.6 \\ \hline 1.6 \end{array} = \begin{array}{c|c} 1.6 \\ \hline 3.2 \end{array}$$

# Basic array operations

$$\text{data} + \text{ones} = \begin{array}{c} \text{data} \\ \begin{array}{|c|c|} \hline 1 & 2 \\ \hline 3 & 4 \\ \hline \end{array} \end{array} = \begin{array}{c} \text{ones} \\ \begin{array}{|c|c|} \hline 1 & 1 \\ \hline 1 & 1 \\ \hline \end{array} \end{array} = \begin{array}{|c|c|} \hline 2 & 3 \\ \hline 4 & 5 \\ \hline \end{array}$$

$$\text{data} + \text{ones\_row} = \begin{array}{c} \text{data} \\ \begin{array}{|c|c|} \hline 1 & 2 \\ \hline 3 & 4 \\ \hline 5 & 6 \\ \hline \end{array} \end{array} + \begin{array}{c} \text{ones\_row} \\ \begin{array}{|c|c|} \hline 1 & 1 \\ \hline \end{array} \end{array} = \begin{array}{c} \text{data} \\ \begin{array}{|c|c|} \hline 1 & 2 \\ \hline 3 & 4 \\ \hline 5 & 6 \\ \hline \end{array} \end{array} + \begin{array}{c} \text{ones\_row} \\ \begin{array}{|c|c|} \hline 1 & 1 \\ \hline 1 & 1 \\ \hline 1 & 1 \\ \hline \end{array} \end{array} = \begin{array}{|c|c|} \hline 2 & 3 \\ \hline 4 & 5 \\ \hline 6 & 7 \\ \hline \end{array}$$

# More useful array operations

data

1
2
3

$$\text{.max()} = 3$$

data

1
2
3

$$\text{.min()} = 1$$

data

1
2
3

$$\text{.sum()} = 6$$

data

1	2
3	4
5	6

$$\text{.max()} = 6$$

data

1	2
3	4
5	6

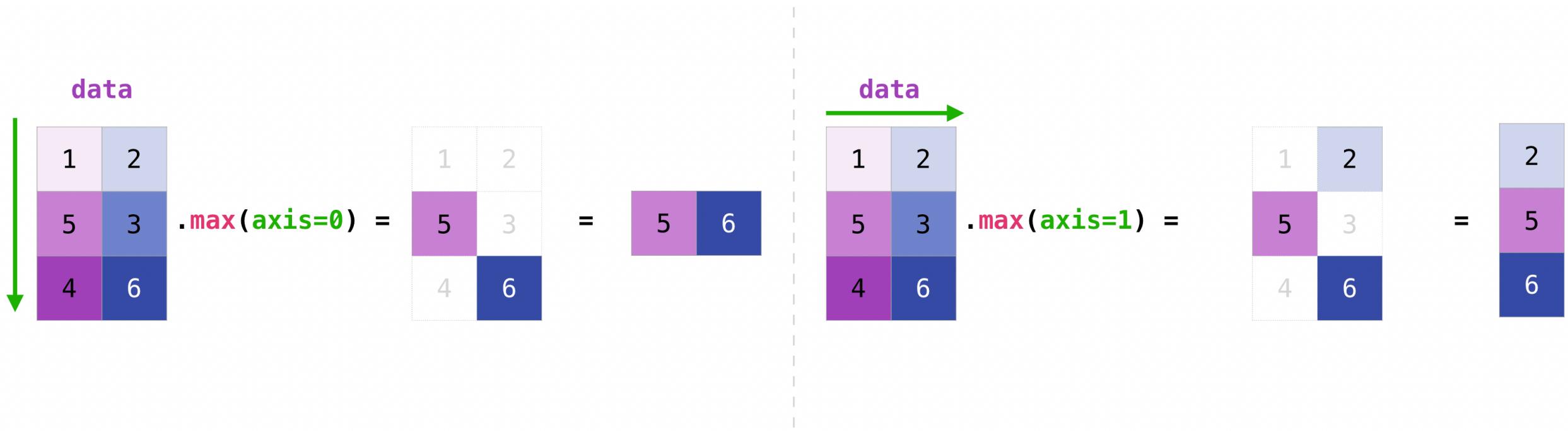
$$\text{.min()} = 1$$

data

1	2
3	4
5	6

$$\text{.sum()} = 21$$

# More useful array operations



# NumPy Copy vs View

## View

`arr[1:4]`



Merely offset is changed

## Copy

`arr[[1,2,3]]`



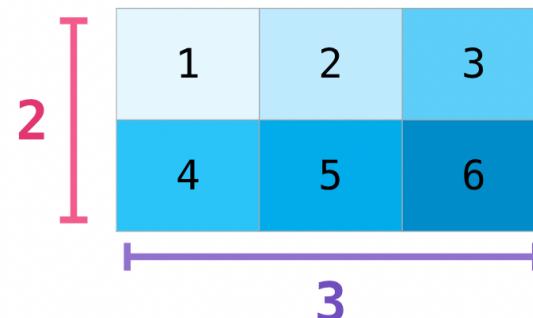
Elements are copied  
and a new object is  
created

# NumPy Array Reshape

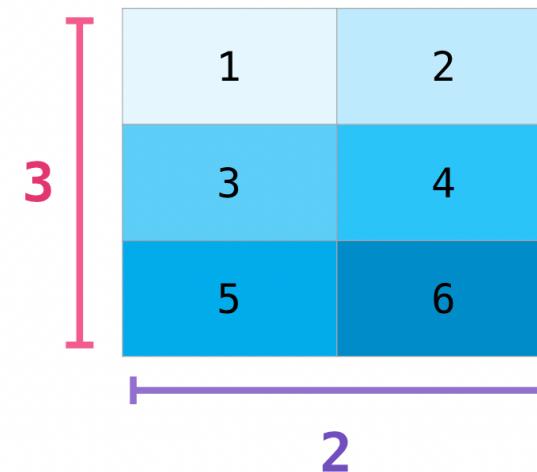
`data`

1
2
3
4
5
6

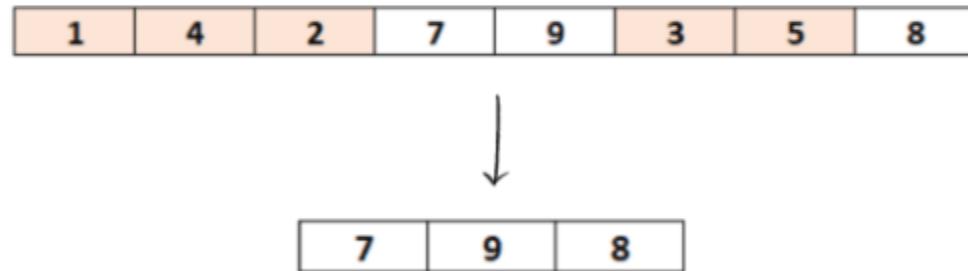
`data.reshape(2,3)`



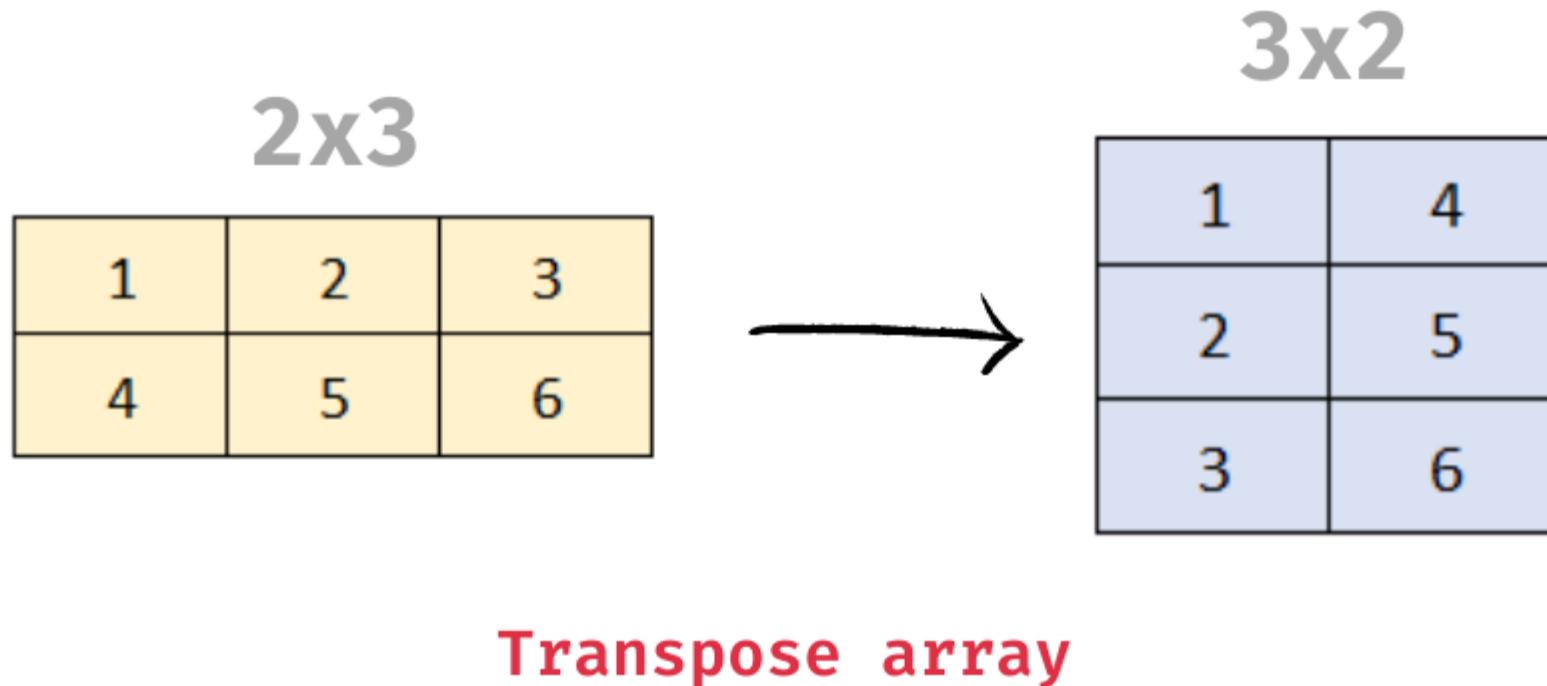
`data.reshape(3,2)`



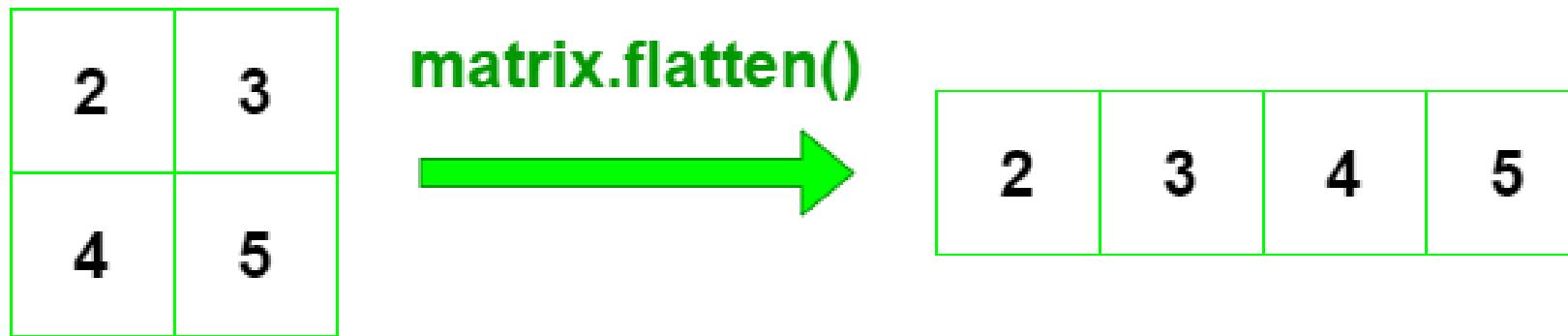
# NumPy Filter Array



# NumPy Arrays Transpose



# flattening multidimensional arrays

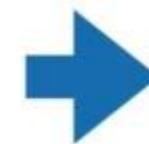


# Dot product vs. element-wise

Element-wise

$$\begin{array}{|c|c|} \hline A & B \\ \hline C & D \\ \hline \end{array} * \begin{array}{|c|c|} \hline E & F \\ \hline G & H \\ \hline \end{array}$$

2x2                    2x2



$$\begin{array}{|c|c|} \hline A^*E & B^*F \\ \hline C^*G & D^*H \\ \hline \end{array}$$

2x2

Dot product

$$\begin{array}{|c|c|c|} \hline A & B & C \\ \hline D & E & F \\ \hline G & H & I \\ \hline \end{array} .\text{dot}(\begin{array}{|c|c|} \hline J & K \\ \hline L & M \\ \hline N & O \\ \hline \end{array})$$

3x3                    3x2



$$\begin{array}{|c|c|} \hline A^*J + B^*L + C^*N & A^*K + B^*M + C^*O \\ \hline D^*J + E^*L + F^*N & D^*K + E^*M + F^*O \\ \hline G^*J + H^*L + I^*N & G^*K + H^*M + I^*O \\ \hline \end{array}$$

3x2

# Numpy Functions

- np.sqrt()
- np.max()
- np.log()
- np.sin()
- np.mean()
- np.square()
- np.sum()
- np.sort()
- np.quantile()
- np.percentile
- np.cov
- np.linalg.norm

# Working with mathematical formulas

$$MeanSquareError = \frac{1}{n} \sum_{i=1}^n (Y_{prediction_i} - Y_i)^2$$



# Working with mathematical formulas

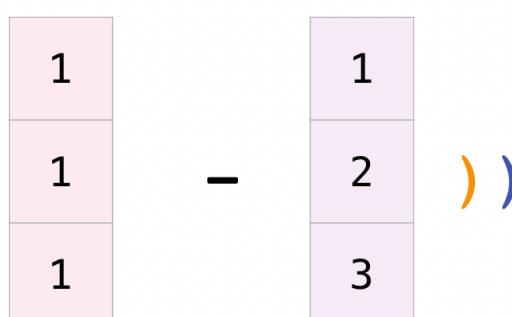
$$MeanSquareError = \frac{1}{n} \sum_{i=1}^n (Y_{prediction_i} - Y_i)^2$$

```
error = (1/n) * np.sum(np.square(predictions - labels))
```

predictions	labels
1	1
1	2
1	3

```
error = (1/3) * np.sum(np.square(
```



# Working with mathematical formulas

```
error = (1/3) * np.sum(np.square(
```

0
-1
-2

```
) )
```

```
error = (1/3) * np.sum(
```

0
1
4

```
)
```

```
error = (1/3) * 5
```

# Calculate Cosine Similarity

```
# import required libraries
import numpy as np
from numpy.linalg import norm

# define two lists or array
A = np.array([2,1,2,3,2,9])
B = np.array([3,4,2,4,5,5])

# compute cosine similarity
cosine = np.dot(A,B)/(norm(A)*norm(B))
print("Cosine Similarity:", cosine)
```

```
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np

array_vec_1 = np.array([[12, 41, 60, 11, 21]])
array_vec_2 = np.array([[40, 11, 04, 11, 14]])

cosine_similarity(array_vec_1 ,
array_vec_2)
```

**Write a NumPy program to find a matrix or vector norm.**

```
import numpy as np
v = np.arange(7)
result = np.linalg.norm(v)
print("Vector norm:")
print(result)
m = np.matrix('1, 2; 3, 4')
result1 = np.linalg.norm(m)
print("Matrix norm:") print(result1)
```

**Write a NumPy program to compute the multiplication of two given matrixes.**

```
import numpy as np
p = [[1, 0], [0, 1]]
q = [[1, 2], [3, 4]]
print("original matrix:")
print(p)
print(q)
result1 = np.dot(p, q)
print("Result of the said matrix
multiplication:")
print(result1)
```



**Write a NumPy program to compute the 80th percentile for all elements in a given array**

**Write a NumPy program to compute IQR for all elements in a given array**

**Compute the mean, standard deviation, and variance of a given array**

**Compute the covariance matrix of two given arrays**



# Data Quality

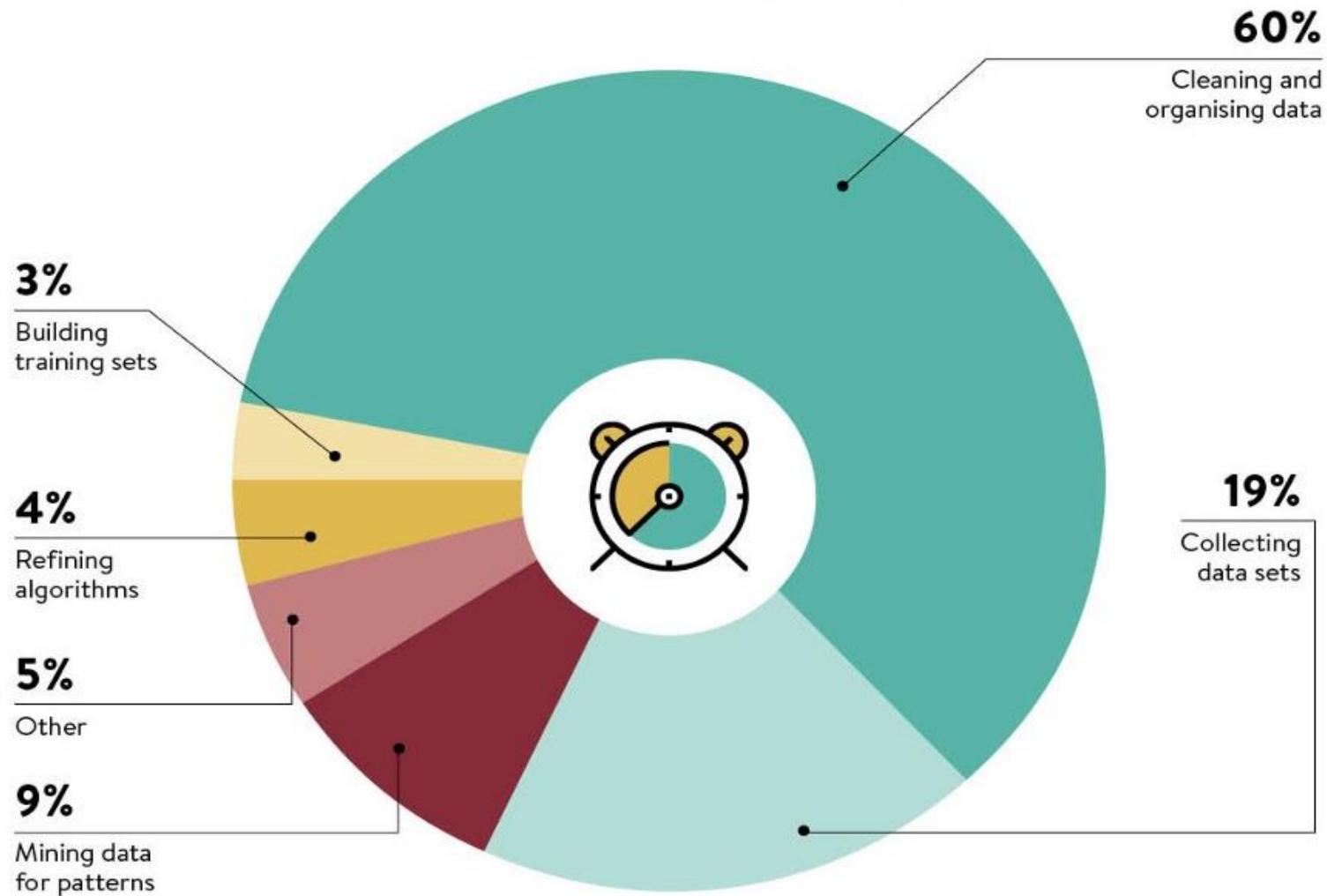
# Data Quality

The future belongs to those who can control their data

Q =  
uality



# Data Quality



Source: CrowdFlower 2016

# Data Quality



# Data Quality



# Data Quality

System A

Customer	Age	Zip Code	City	Rate
John Smith		95113	San José	1
Bob Clark	41	10007	New York	2
Jane Harris	-10	60602	Chicago	3

Completeness: Missing data

Validity: Is this a valid age?

System B

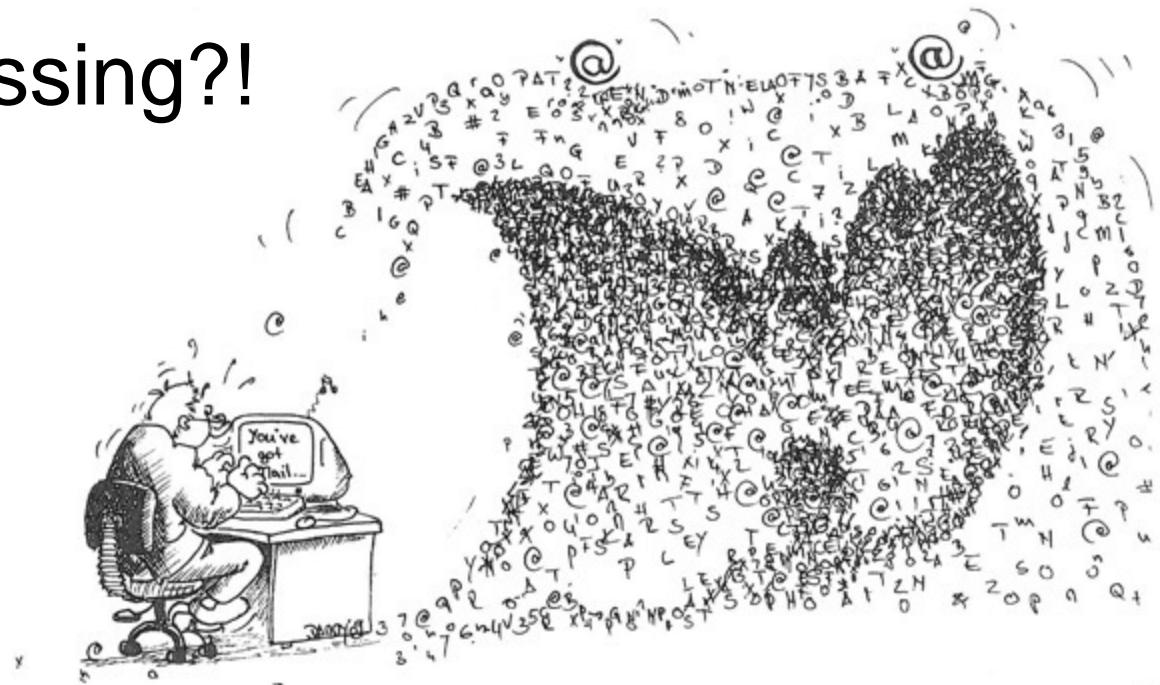
Customer	Age	Zip Code	City	Rate
John Smitz	15	95113	San José	A
Robert Clark	30	17000	New York	B
Mark Lee	32	60602	Chicago	C

Integrity: same city, different zip codes

The diagram illustrates data quality issues across two systems, A and B. In System A, the 'Age' field for Jane Harris contains the invalid value '-10'. In System B, the 'Zip Code' field for Robert Clark contains the invalid value '17000'. Both the 'City' fields in both systems contain the value 'New York', which is circled in orange, indicating a potential integrity issue where multiple records share the same city name despite having different zip codes.

# Dealing with Missing Values

- Take a first look at the data.
- How many missing data points do we have?
- Figure out why the data is missing!?



# Strategies To Handle Missing Values

**Missing values**

PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	male	22	1	0	A/5 21171	7.25		S
2	1	1	female	38	1	0	PC 17599	71.2033	C85	C
3	1	3	female	26	0	0	STON/O2. 3101282	7.925		S
4	1	1	female	35	1	0	113803	53.1	C123	S
5	0	3	male	35	0	0	373450	8.05		S
6	0	3	male		0	0	330877	8.4583		Q

# Strategies To Handle Missing Values

- Remove observation/records that have missing values.
- Imputation.

# Deleting Columns & Rows with Missing Values

- Here, we either delete a particular row if it has a null value for a particular feature and a particular column if it has more than **70-75%** of missing values.
- This method is advised **only** when there are **enough samples** in the data set.
- Removing the data will lead to **loss of information** which will not give the expected results while predicting the output.

# Handling Continuous variables

- This strategy can be applied on a feature which has **numeric** data like the age of a person.
- We can calculate the **mean, median or mode** of the feature and replace it with the missing values.
- This is an **approximation** which **can add variance** to the data set.
- But the loss of the data can be negated by this method which yields **better results** compared to removal of rows and columns.

# Handling categorical variables

- A **categorical** feature will have a definite number of possibilities, such as gender, for example.
- Since they have a definite number of classes, we can **assign another class** for the missing values.

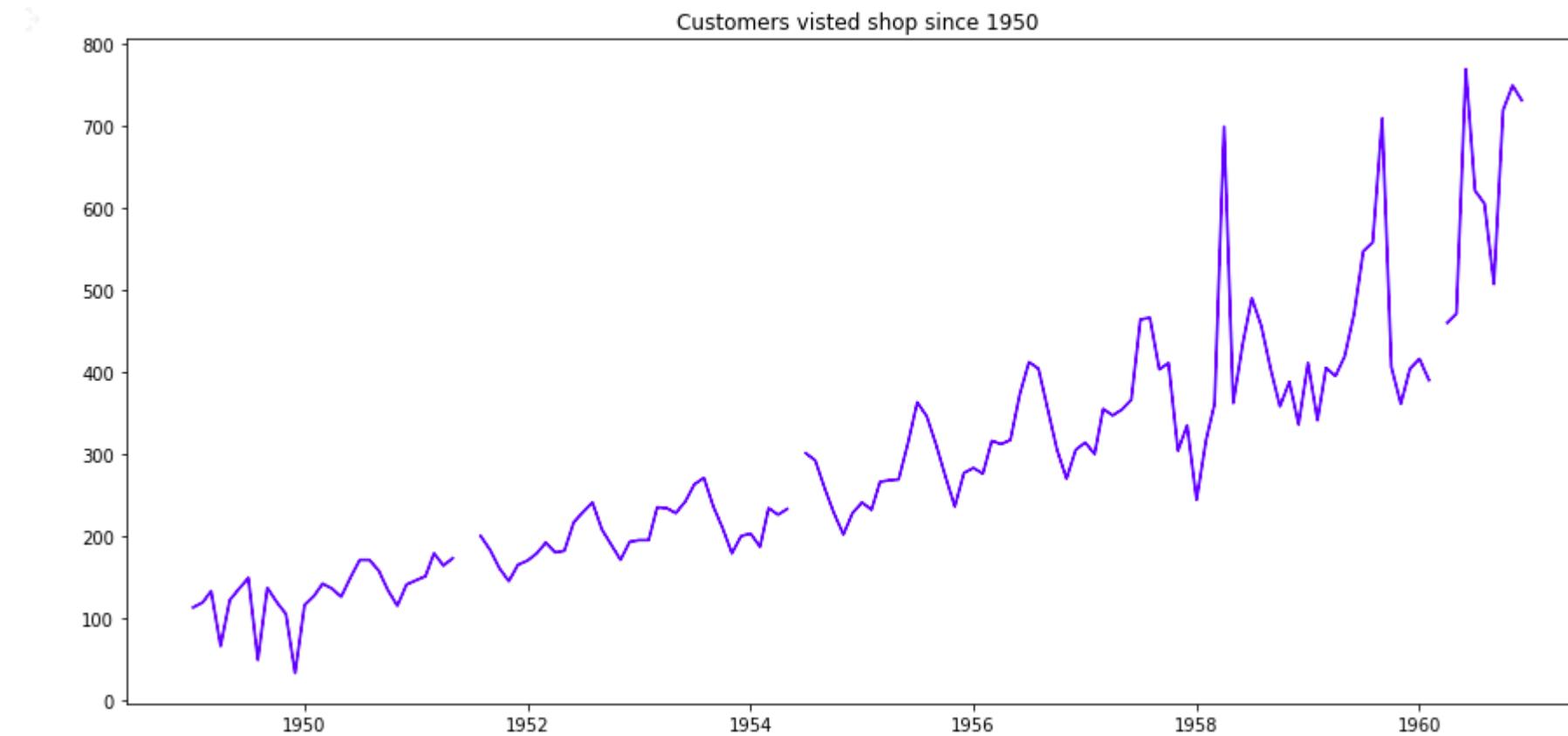
# Predicting The Missing Values

- Using the features which do not have missing values, we can predict the nulls with the help of a machine learning algorithm.
- This method may result in **better accuracy**, unless a missing value is expected to have a very **high variance**.
  - Example: linear regression to replace the nulls in the feature ‘age’, using other available features.

# Using Algorithms Which Support Missing Values

- KNN is a machine learning algorithm which works on the principle of distance measure.
- This algorithm can be used when there are nulls present in the dataset.
- While the algorithm is applied, KNN considers the missing values by taking the majority of the K nearest values.
- In this particular dataset, taking into account the person's age, sex, class etc, we will assume that people having same data for the above mentioned features will have the same kind of fare.
- Another algorithm which can be used here is **RandomForest**.
- This model produces a robust result because it works well on **non-linear** and the **categorical** data.

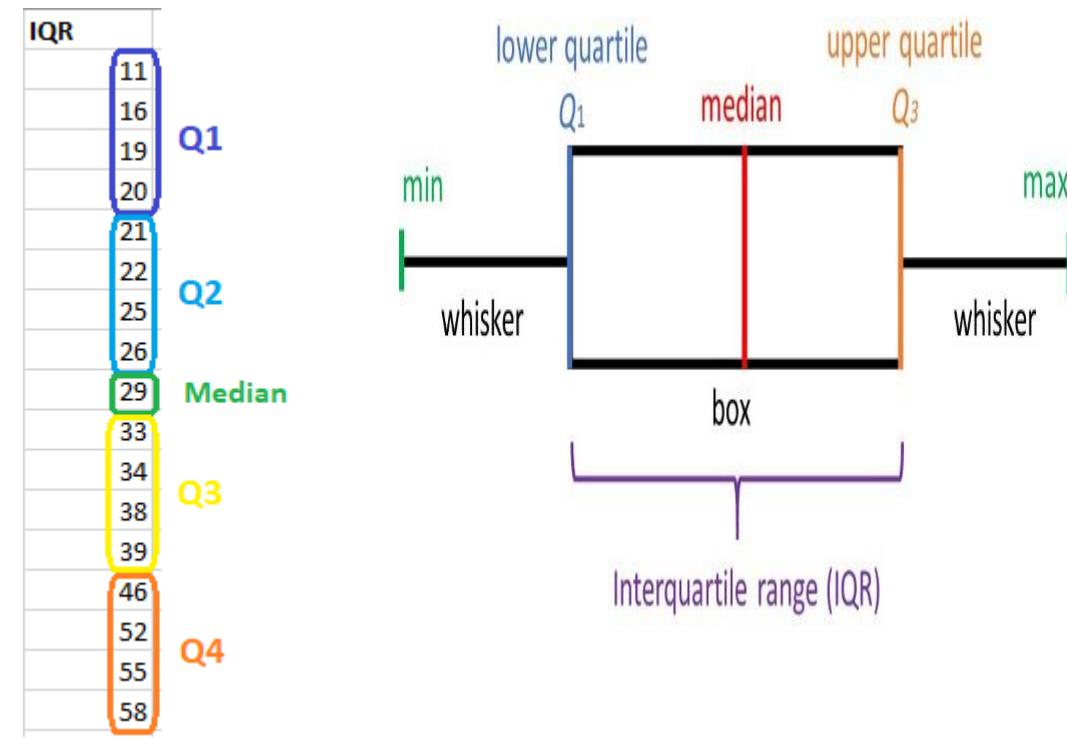
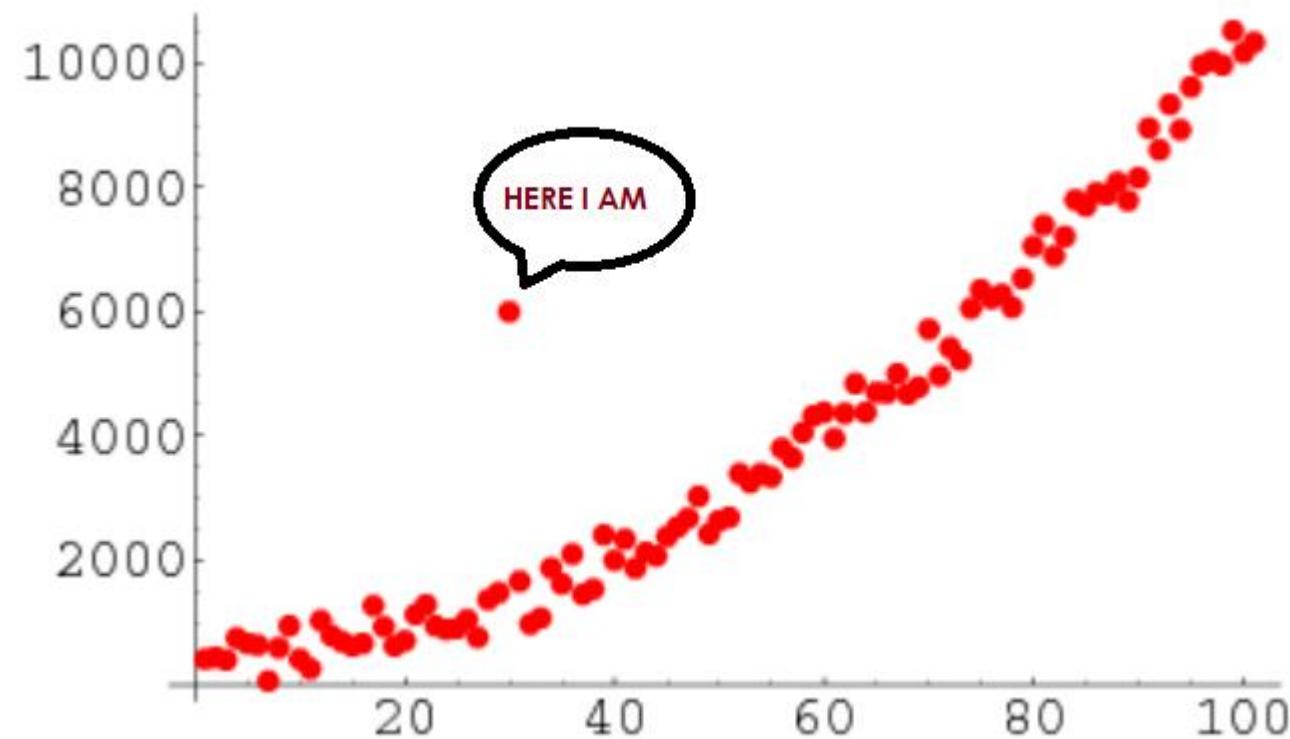
# Time Series Imputations



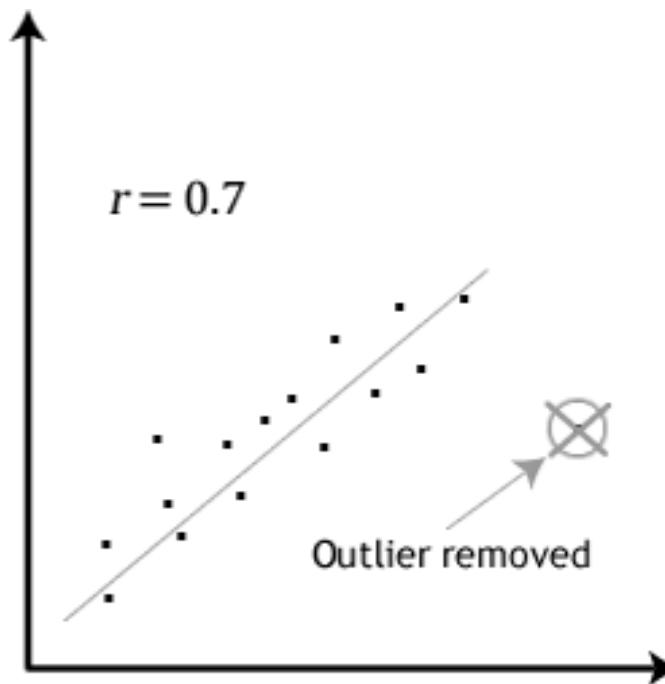
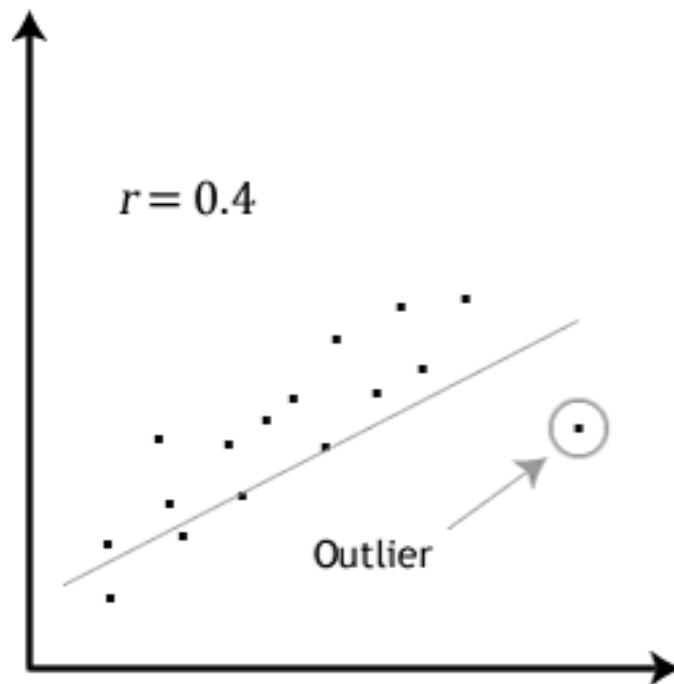
**Interpolation** is the process of estimating and inserting missing values in time series data.

# Outliers

- An **outlier** is an observation that lies an abnormal distance from other values in a random sample from a population.

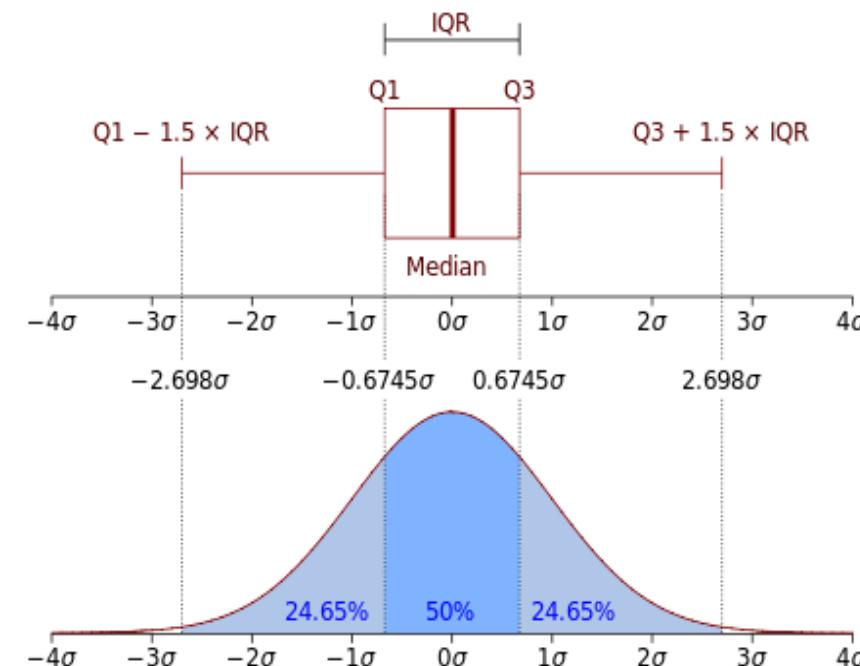


# Impact Of Outliers On Linear Regression



# How To Identify Outliers?

- Box Plot
- Histogram
- Scatter Plot
- Interquartile Range (IQR) based method
- Standard Deviation based method



# Exercise

Find  $Q_1$ ,  $Q_2$ , and  $Q_3$  for the following data set. Identify any outliers, and draw a box-and-whisker plot.

$$\{5, 40, 42, 46, 46, 48, 49, 50, 50, 52, 53, 55, 56, 58, 75, 102\}$$

There are 15 values, arranged in increasing order. So,  $Q_2$  is the 8<sup>th</sup> data point, 50.

$Q_1$  is the 4<sup>th</sup> data point, 46, and  $Q_3$  is the 12<sup>th</sup> data point, 56.

The interquartile range IQR is  $Q_3 - Q_1$  or  $56 - 46 = 10$ .

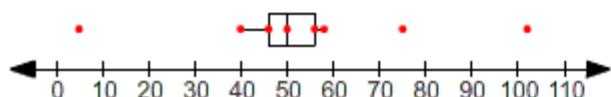
Now we need to find whether there are values less than  $Q_1 - (1.5 \times \text{IQR})$  or greater than  $Q_3 + (1.5 \times \text{IQR})$ .

$$Q_1 - (1.5 \times \text{IQR}) = 46 - 15 = 31$$

$$Q_3 + (1.5 \times \text{IQR}) = 56 + 15 = 71$$

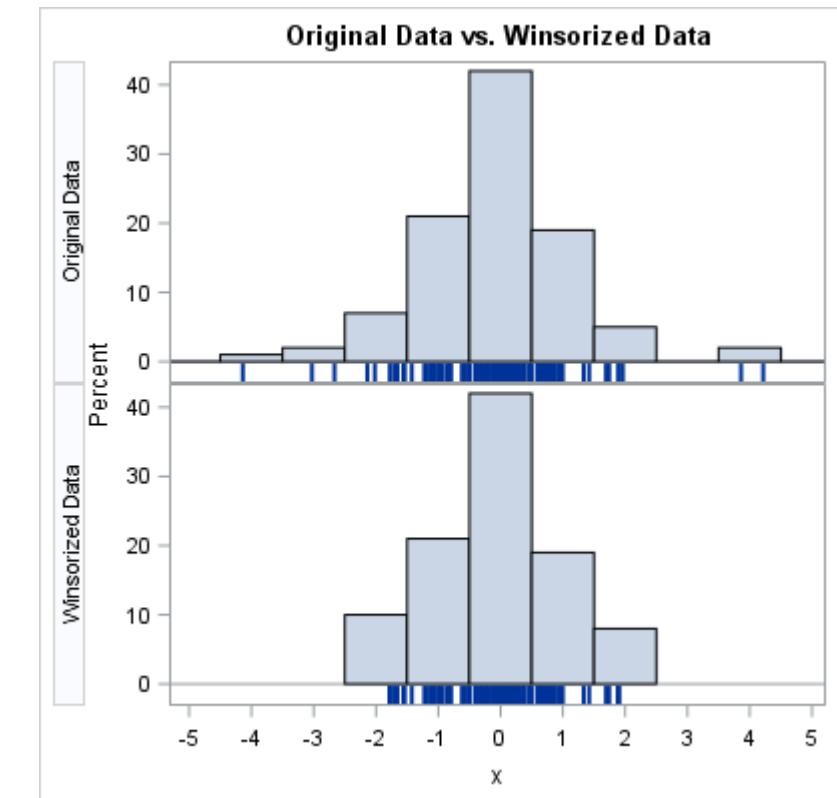
Since 5 is less than 31 and 75 and 102 are greater than 71, there are 3 outliers.

The box-and-whisker plot is as shown. Note that 40 and 58 are shown as the ends of the whiskers, with the outliers plotted separately.



# How To Handle The Outliers?

- Doing nothing
- Deleting/Trimming
- Winsorizing (replace the extreme values with a certain percentile value)
- Transformation
- Binning
- Use robust estimators
- Imputing



- List of Machine Learning algorithms which are sensitive to outliers:  
① Linear Regression ② Logistic Regression ③ Support Vector Machine ④ K- Nearest Neighbors ⑤ K-Means Clustering ⑥ Hierarchical Clustering ⑦ Principal Component Analysis ⑧ AdaBoost
- List of Machine Learning algorithms which are `not` sensitive to outliers: ① Decision Tree ② Random Forest ③ XGBoost ④ Naive Bayes ⑤ DBSCAN ⑥ NN

# Pandas

---

# Pandas Agenda



# Pandas

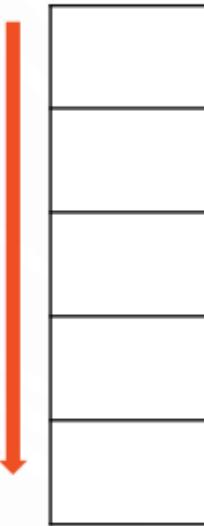
- Pandas is an open source library built on top of NumPy
- It allows for fast analysis and data cleaning and preparation
- It excels in performance and productivity.
- It also has built-in visualization features.
- It can work with data from a wide variety of sources.

# Pandas Data Structure

## One-dimensional data structure

- contains values along **a single** axis (rows)

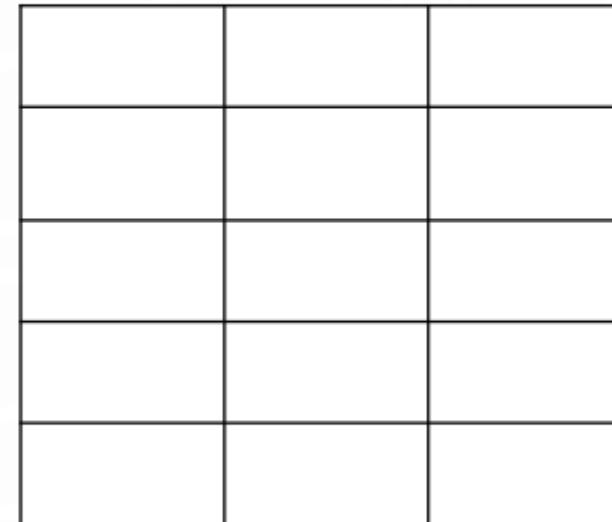
rows



**Series**

## Two-dimensional data structure

columns

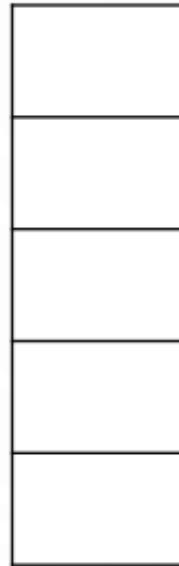


**DataFrame**

# Pandas Data Structure

## Single column data

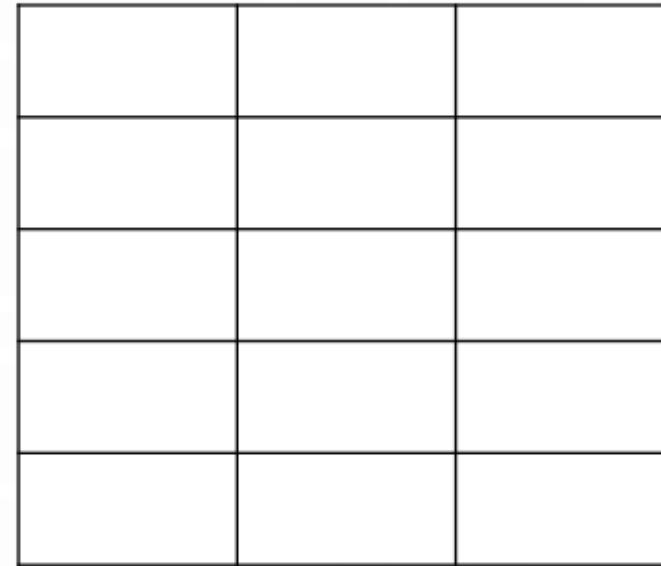
- o corresponds to a **single variable**
- o information of a **single type**



**Series**

## Multi-column data

- o each column represents a **different variable**
- o every column contains data of **its own type**
- o the information can potentially be **heterogeneous**



**DataFrame**

# Pandas Data Structure

	Name	Team	Number
0	Avery Bradley	Boston Celtics	0.0
1	John Holland	Boston Celtics	30.0
2	Jonas Jerebko	Boston Celtics	8.0
3	Jordan Mickey	Boston Celtics	NaN
4	Terry Rozier	Boston Celtics	12.0
5	Jared Sullinger	Boston Celtics	7.0
6	Evan Turner	Boston Celtics	11.0

ser = pd.Series (df [ 'Name'])

ser = pd.Series (df [ 'Team'])

ser = pd.Series (df [ 'Number'])

# Creating a Series

**pd.Series(data=None, index=None)**

```
# simple list
lst = ['G', 'E', 'E', 'K', 'S', 'F', 'O', 'R', 'G', 'E', 'E', 'K', 'S']
```

```
# forming series
s = pd.Series(lst)
```

```
# simple dict
dct = {'G':2, 'E':4, 'K':2, 'S':2, 'F':1, 'O':1, 'R':1}
```

```
# forming series
s = pd.Series(dct)
```

```
# numpy array
arr = np.array(['G', 'E', 'E', 'K', 'S', 'F', 'O', 'R', 'G', 'E', 'E', 'K', 'S'])
```

```
# forming series
s = pd.Series(arr)
```

# Indexing & Slicing A Series

Series1

	Series1
A	1
B	2
C	3
D	4

`series1['A']`

`series1[0]`

`series1[0:3]`

`series1[0:5:2]`

`series1['A':'C']`

# Arithmetic Operation

A	1
B	2
C	3
D	4

+

A	1
C	2
E	3
F	4

=

A	2
B	NaN
C	5
D	NaN
E	NaN
F	NaN

# Pandas Data Structure

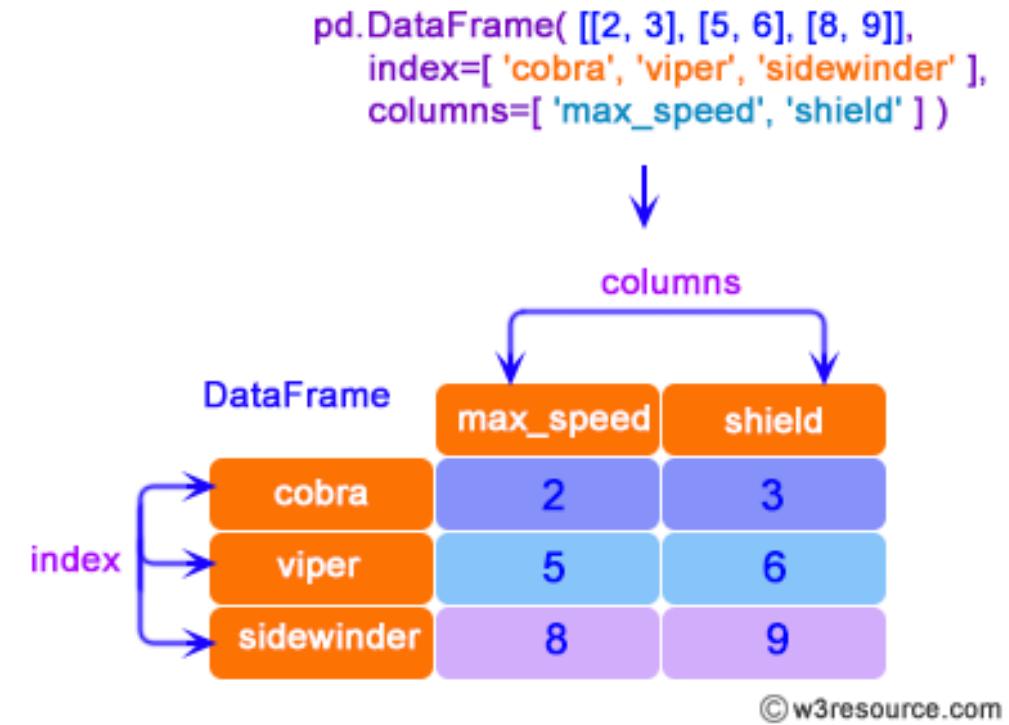
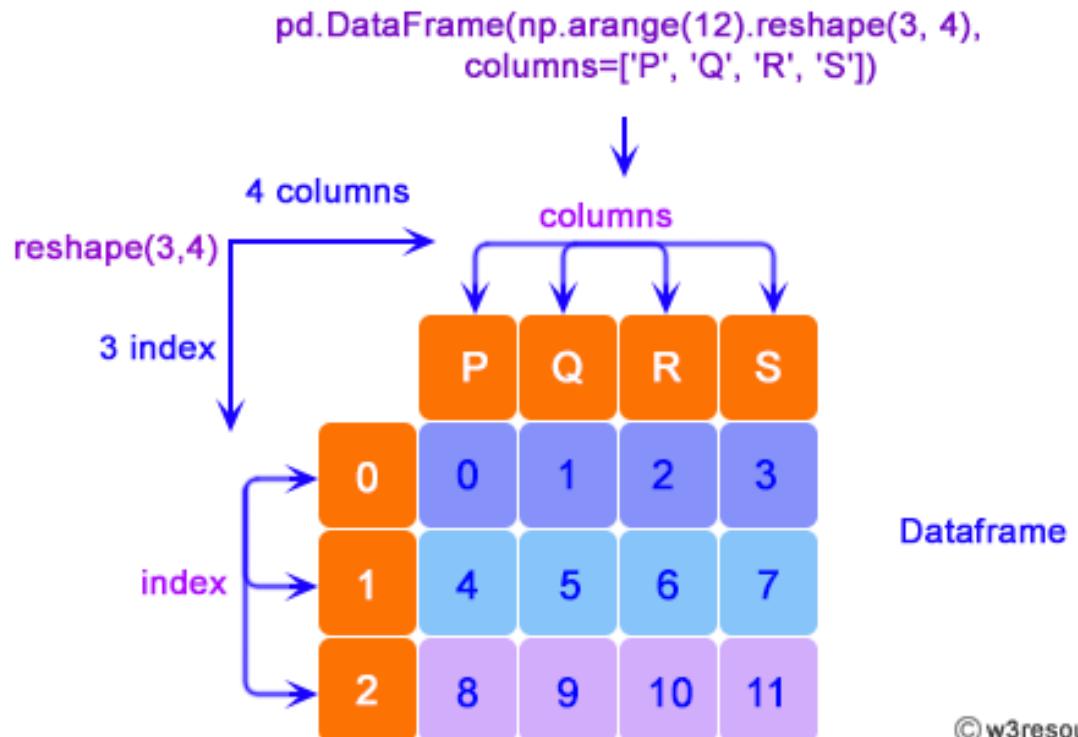
Diagram illustrating the structure of a Pandas DataFrame:

The DataFrame has the following structure:

- Column names:** Name, Team, Number, Position, Age, Height, Weight, College, Salary.
- Index label:** The index is labeled by row number (0 to 6).
- Columns axis=1:** The columns represent different data types: Name (string), Team (string), Number (float), Position (string), Age (float), Height (string), Weight (float), College (string), and Salary (float).
- Index axis=0:** The index represents the position of each row.
- Data:** The data consists of 7 rows of information for basketball players.
- Missing value:** A cell containing "NaN" is highlighted, indicating a missing value.

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
1	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston University	NaN
2	Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0
3	Jordan Mickey	Boston Celtics	NaN	PF	21.0	6-8	235.0	LSU	1170960.0
4	Terry Rozier	Boston Celtics	12.0	PG	22.0	6-2	190.0	Louisville	1824360.0
5	Jared Sullinger	Boston Celtics	7.0	C	NaN	6-9	260.0	Ohio State	2569260.0
6	Evan Turner	Boston Celtics	11.0	SG	27.0	6-7	220.0	Ohio State	3425510.0

# Creating a DataFrame



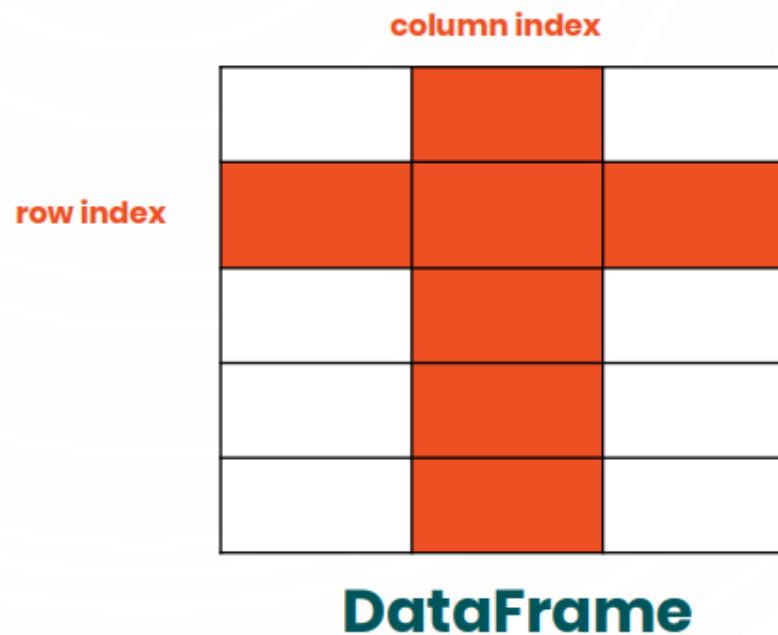
# Creating a DataFrame

```
# Named Index
fruit = {
    'oranges' : [3,2,0,1],
    'apples'  : [0,3,7,2],
    'grapes'  : [5,6,9,0],
    'pear'    : [1,23,45,1]
}
df = pd.DataFrame(fruit ,index = ['June','July','August','September'])
df
```

	oranges	apples	grapes	pear
June	3	0	5	1
July	2	3	6	23
August	0	7	9	45
September	1	2	0	1

# Indexing & Slicing Dataframe

Two points  
of reference



# loc Vs. iloc

`df.loc[ START:STOP:STEP , START:STOP:STEP ]`

Select Rows by Names/Labels

Select Columns by Names/Labels

Select Rows by Names/Labels

Select Columns by Names/Labels

`df.iloc[ START:STOP:STEP , START:STOP:STEP ]`

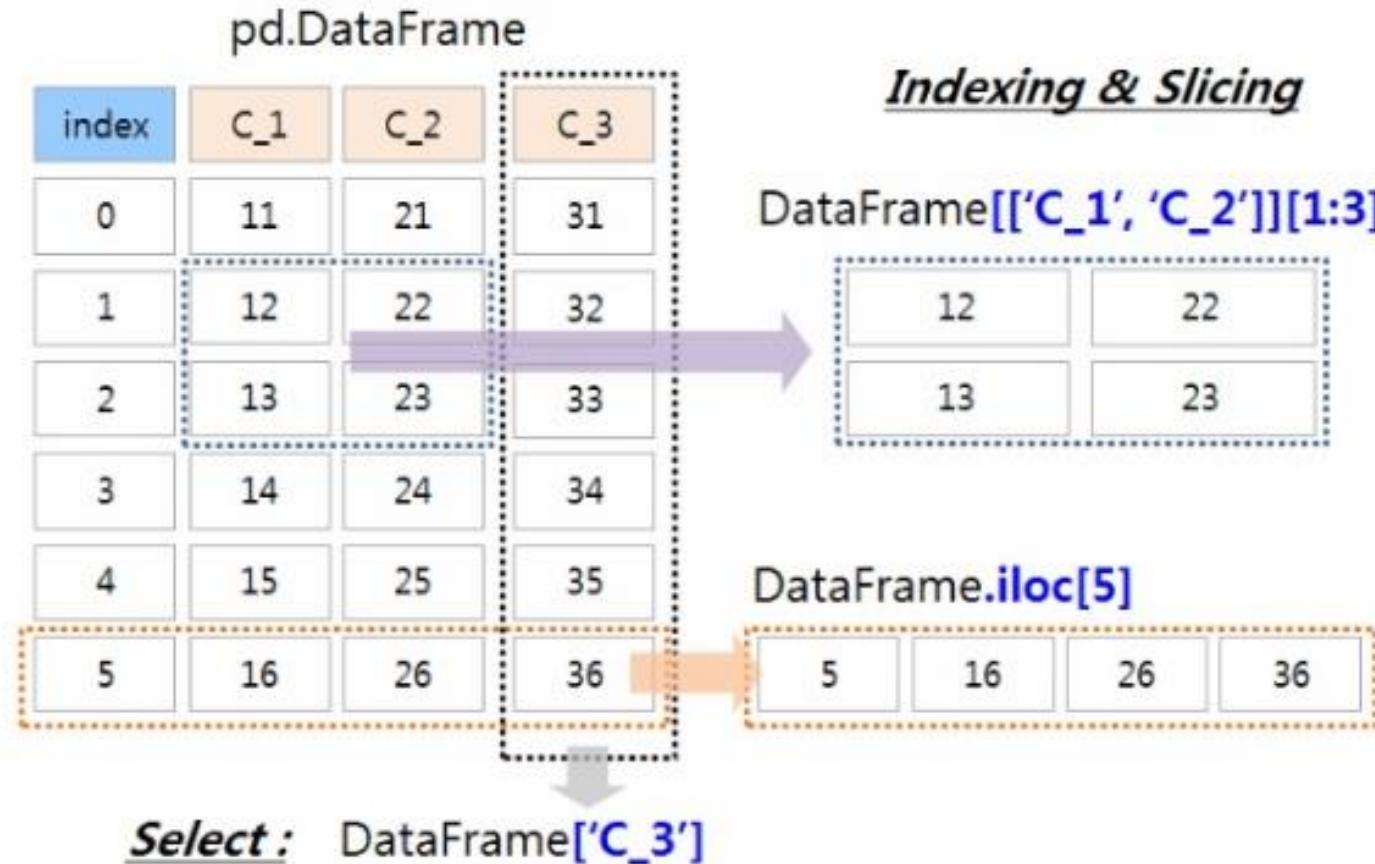
Select Rows by Indexing Position

Select Columns by Indexing Position

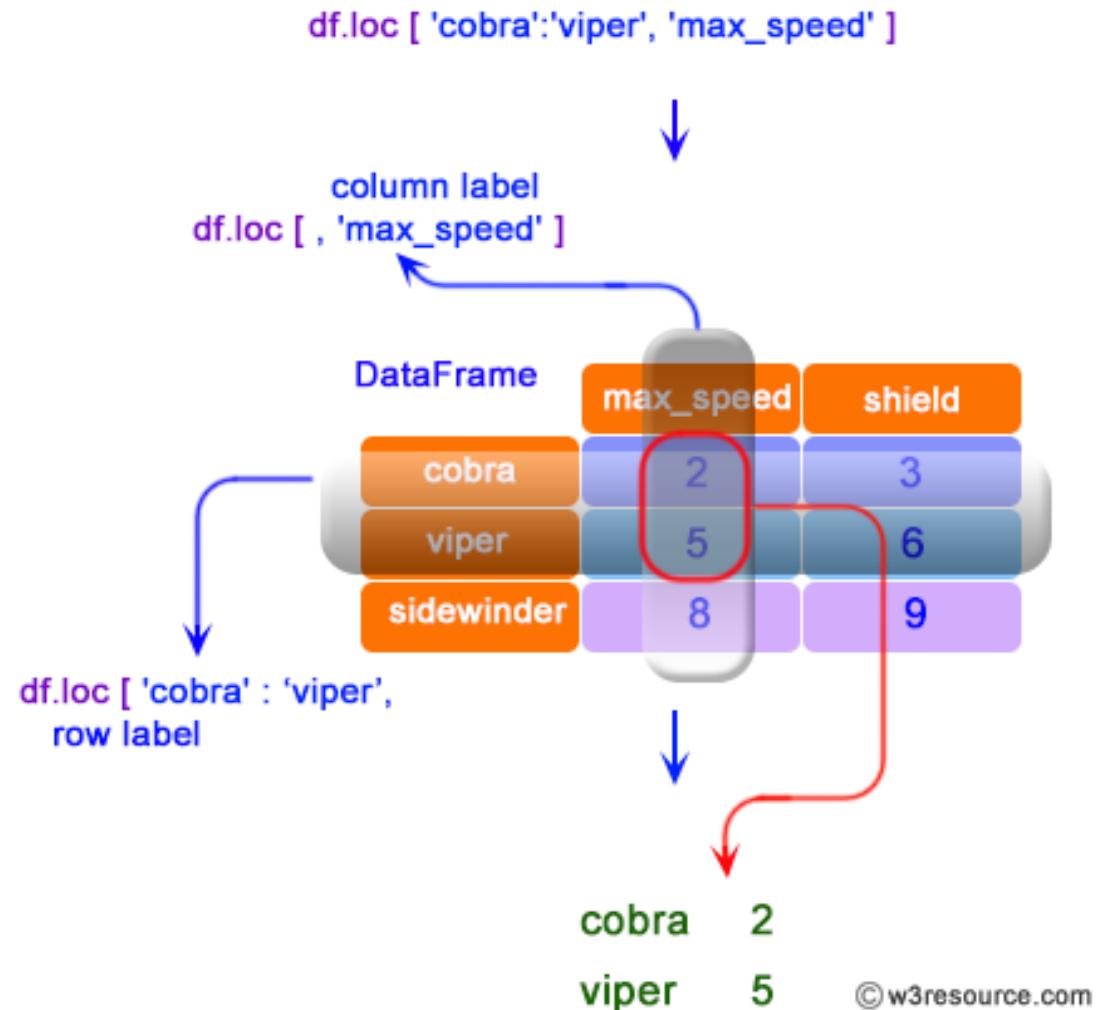
# loc Vs. iloc

	loc[] - By Label	iloc[] - By Index
Select Single Row	df.loc['r2']	df.iloc[1]
Select Single Column	df.loc[:, "Courses"]	df.iloc[:, 0]
Select Multiple Rows	df.loc[['r2','r3']]	df.iloc[[1,2]]
Select Multiple Columns	df.loc[:, ["Courses","Fee"]]	df.iloc[:, [0,1]]
Select Rows Range	df.loc['r1':'r4']	df.iloc[0:4]
Select Columns Range	df.loc[:, 'Fee':'Discount']	df.iloc[:,1:4]
Select Alternate Rows	df.loc['r1':'r4':1]	df.iloc[0:4:1]
Select Alternate Columns	df.loc[:, 'Fee':'Discount':1]	df.iloc[:,1:4:1]

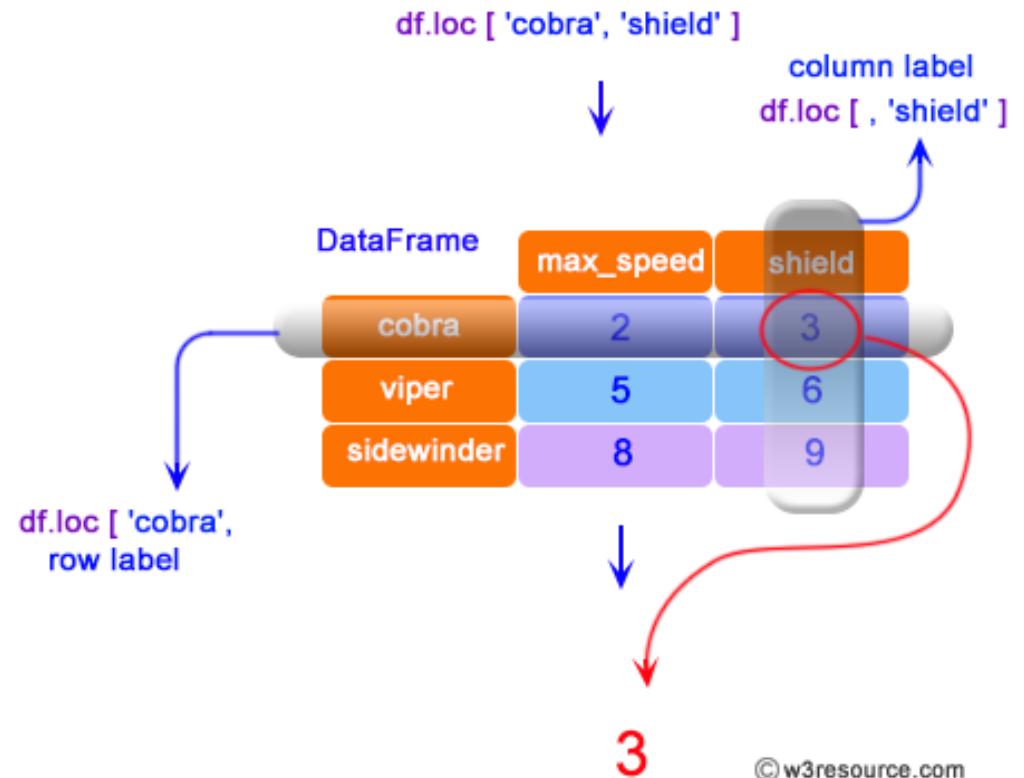
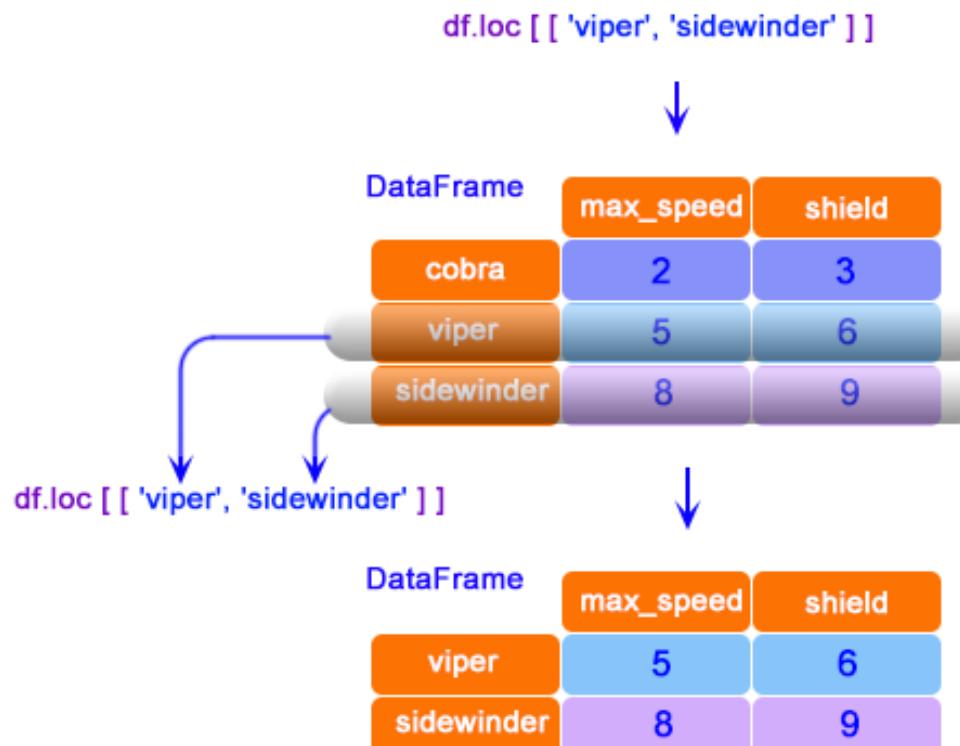
# Indexing and slicing



# loc Vs. iloc

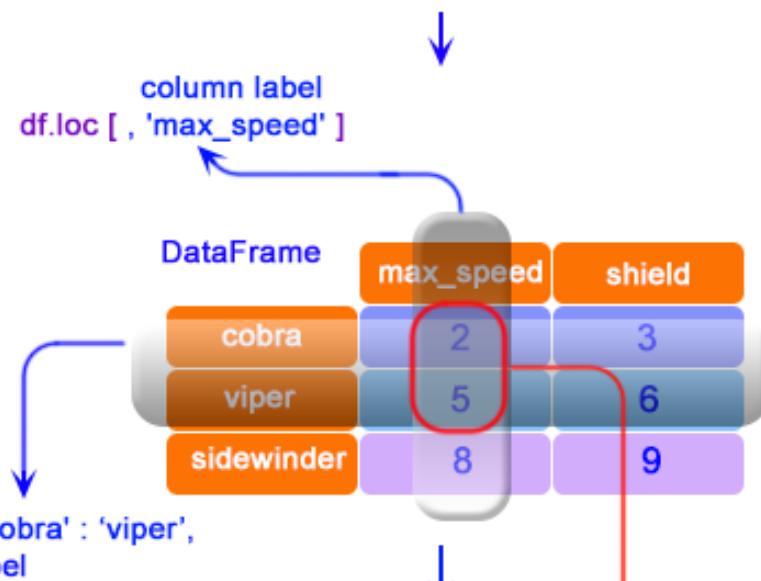


# loc Vs. iloc



# loc Vs. iloc

df.loc [ 'cobra':'viper', 'max\_speed' ]

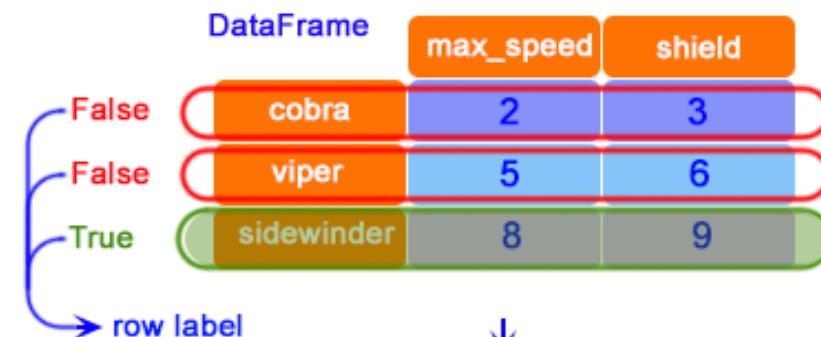


df.loc [ 'cobra' : 'viper',  
row label

cobra 2  
viper 5

© w3resource.com

df.loc [ [ False, False, True ] ]

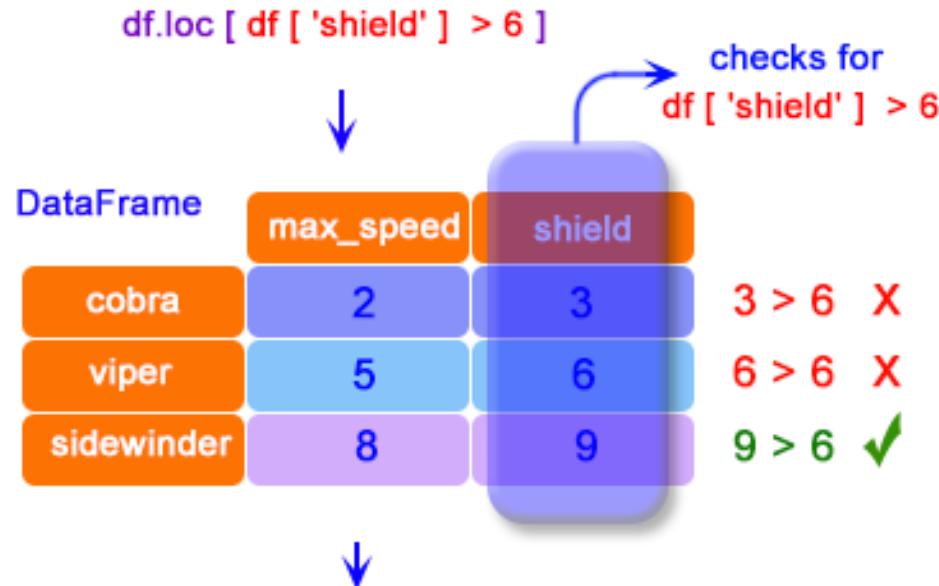


New DataFrame

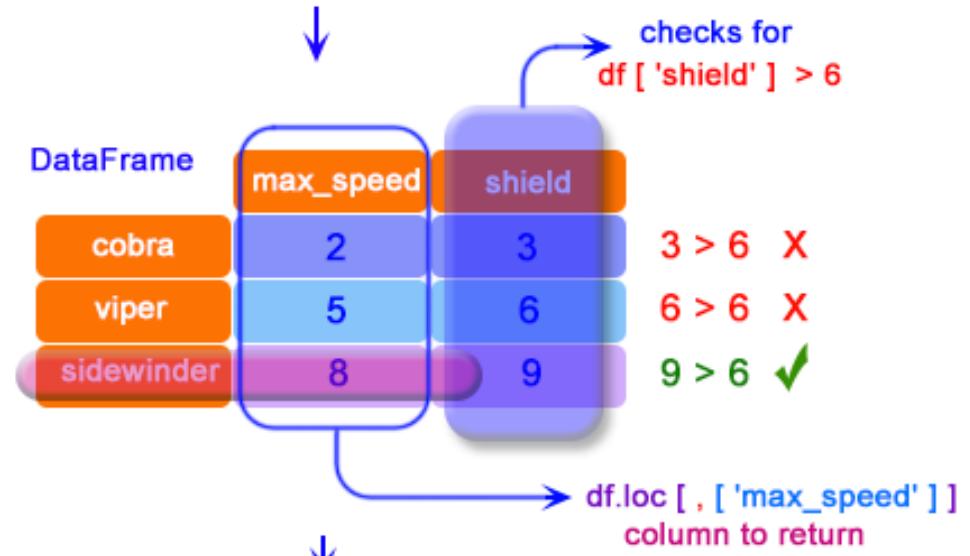
	max_speed	shield
sidewinder	8	9

© w3resource.com

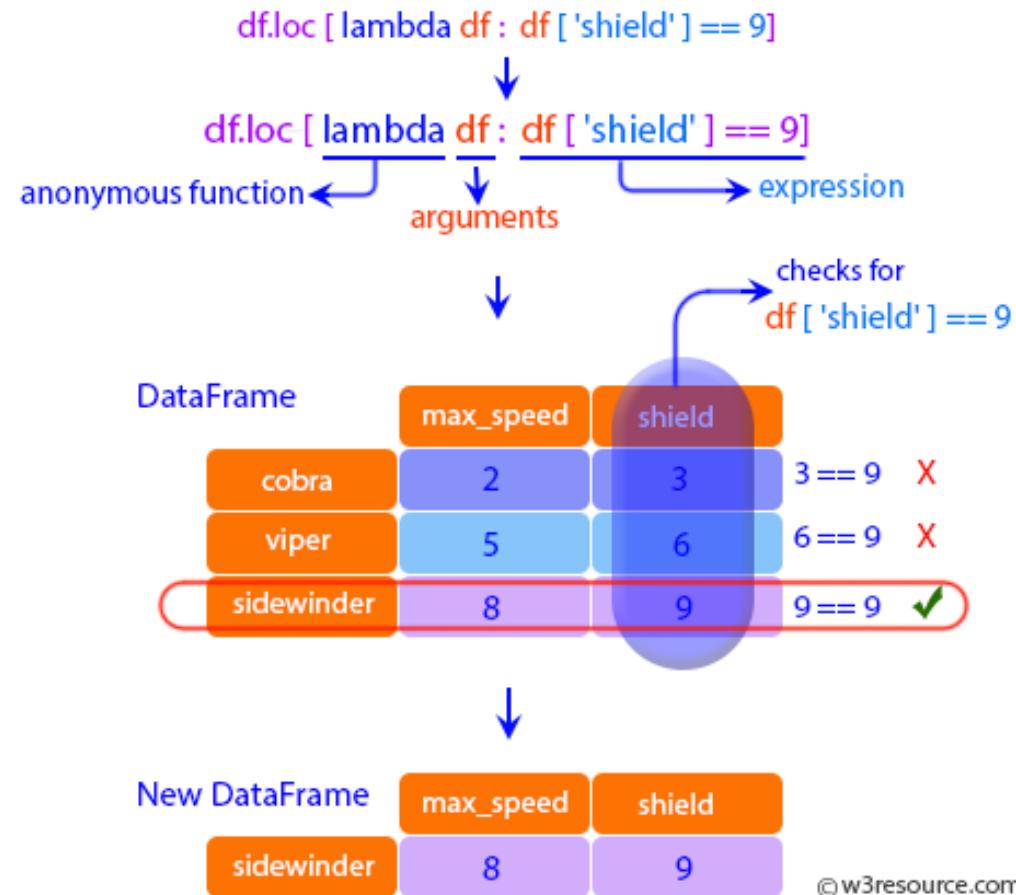
# loc Vs. iloc



df.loc [ df [ 'shield' ] > 6, [ 'max\_speed' ] ]



# loc Vs. iloc



# Filter

```
# filter Rows Based on condition  
df[df["Courses"] == 'Spark']  
df.loc[df['Courses'] == value]  
df.query("Courses == 'Spark'")  
df.loc[df['Courses'] != 'Spark']  
df.loc[df['Courses'].isin(values)]  
df.loc[~df['Courses'].isin(values)]
```

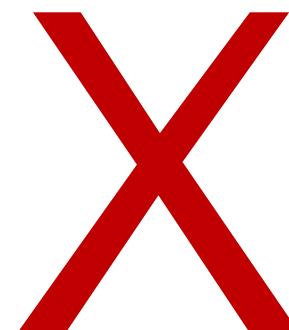
# Filter

```
# filter Multiple Conditions using Multiple Columns  
df.loc[(df['Discount'] >= 1000) & (df['Discount'] <= 2000)]  
df.loc[(df['Discount'] >= 1200) & (df['Fee'] >= 23000 )]
```

# DON'T

df **and** df2

df **or** df2



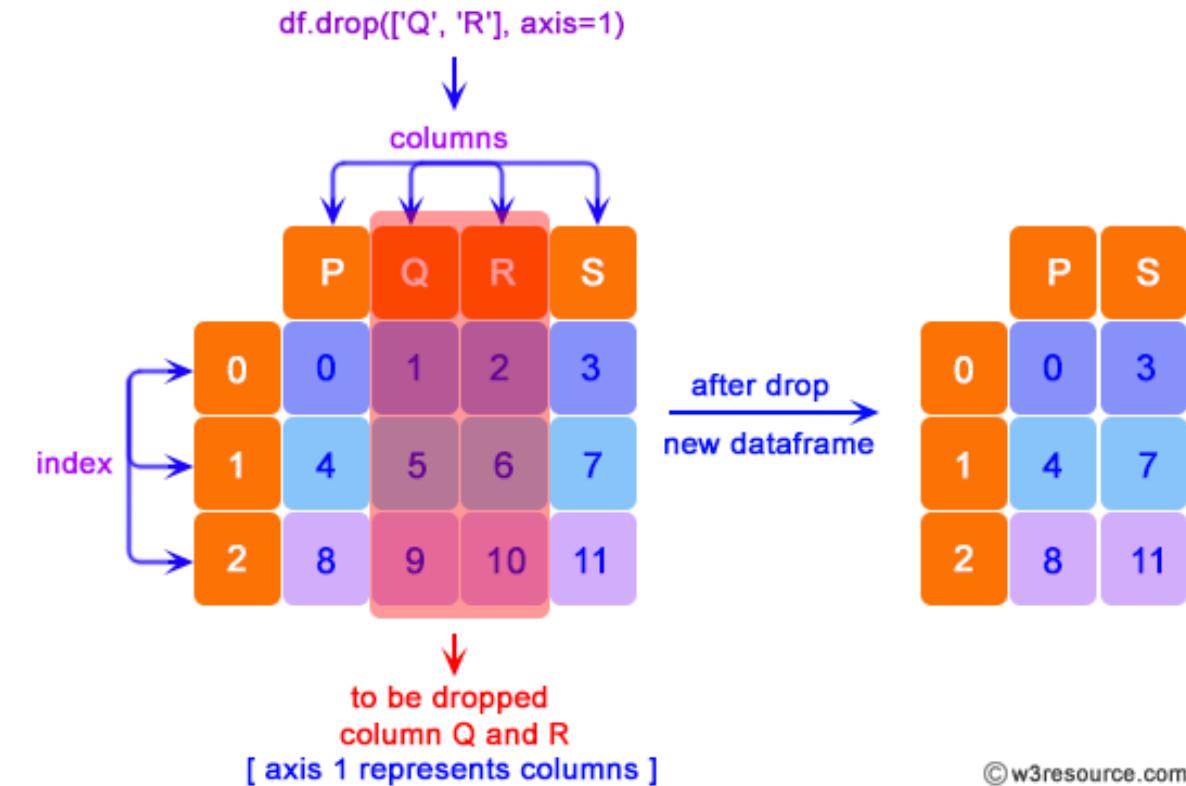
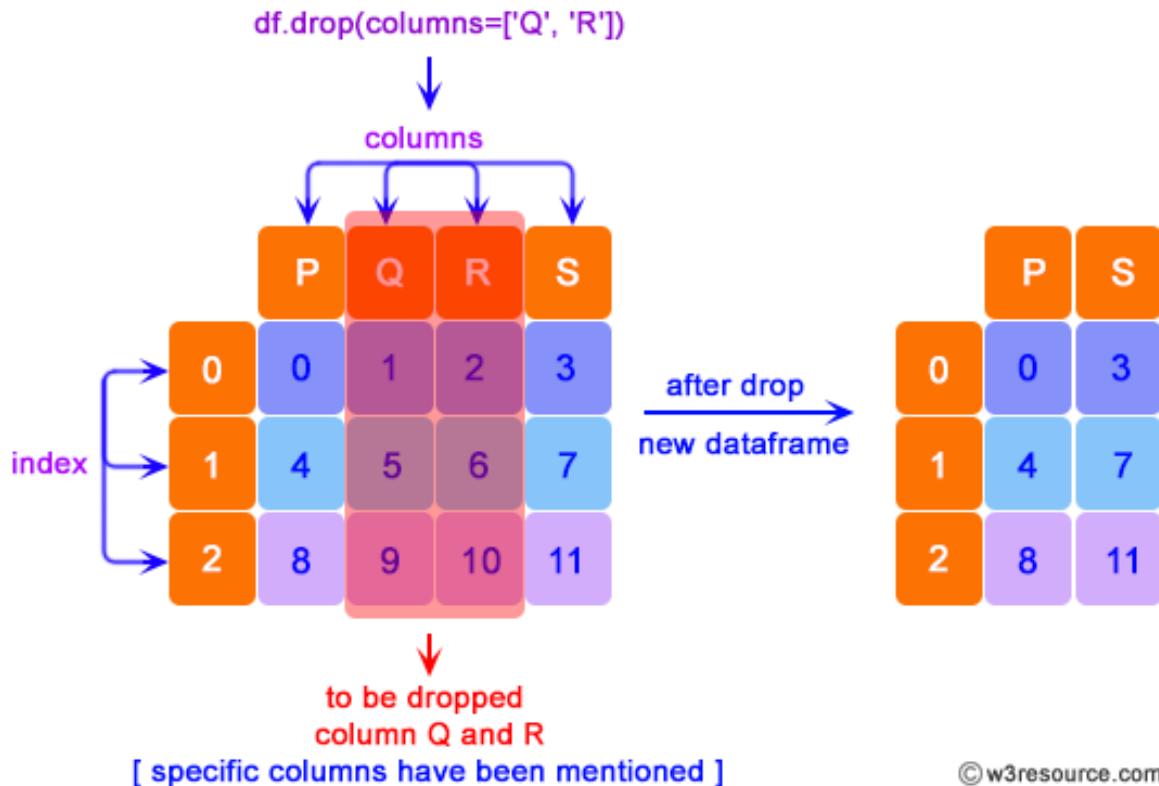
# Filter

```
# Other examples
df[df['Courses'].str.contains("Spark")]
df[df['Courses'].str.lower().str.contains("spark")]
df[df['Courses'].str.startswith("P")]
```

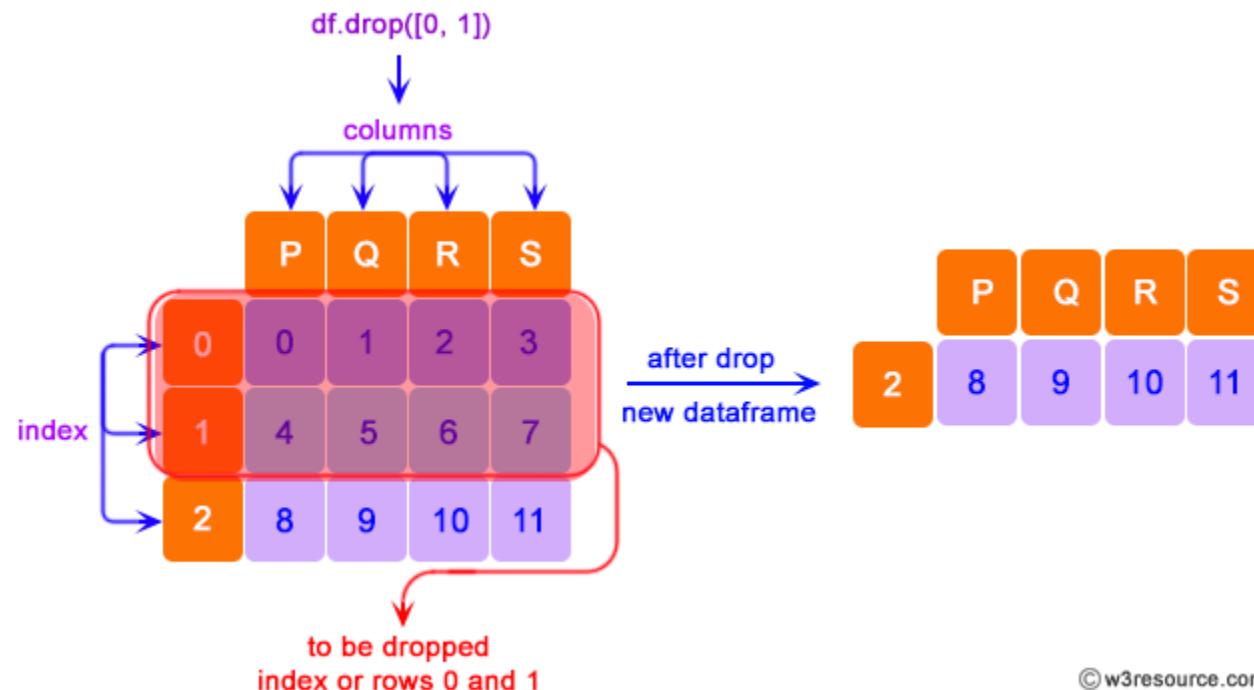
# Analyzing DataFrames

- head()
- tail()
- info()
- describe()

# Drop Function



# Drop Function

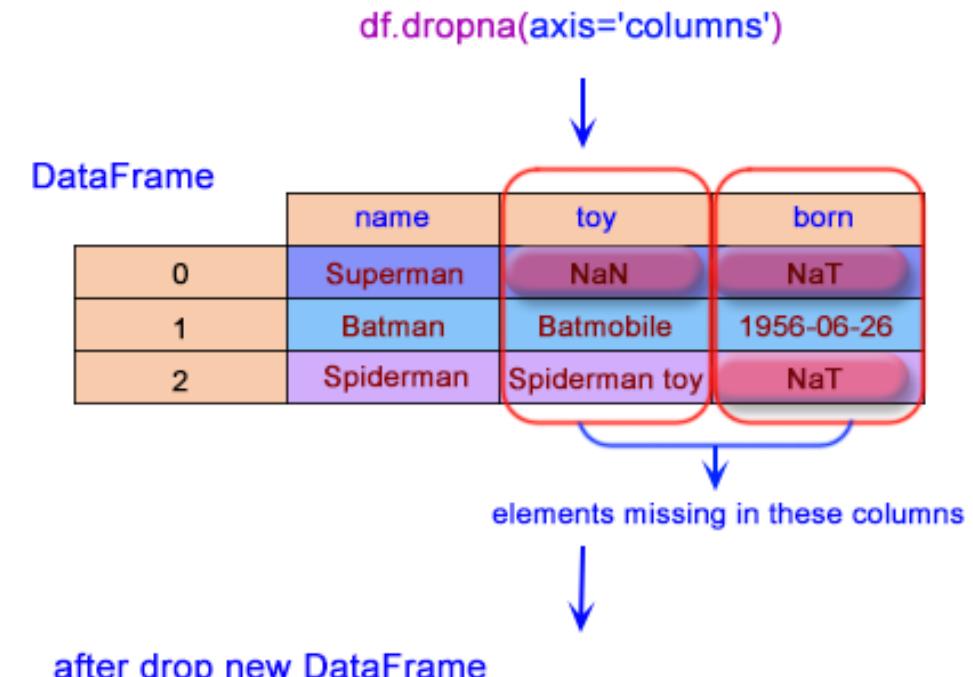
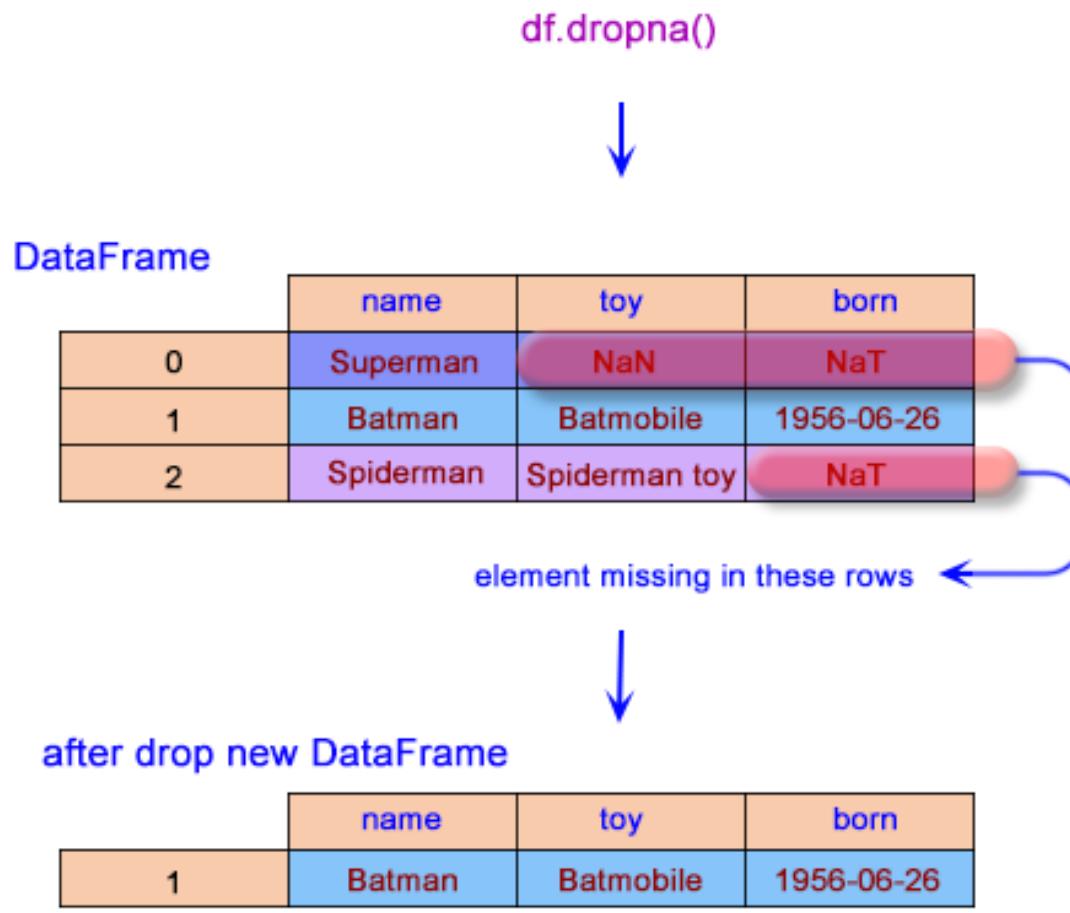


# Cleaning Empty Cells

```
dropna(self, axis=0, how="any", thresh=None, subset=None,  
inplace=False)
```

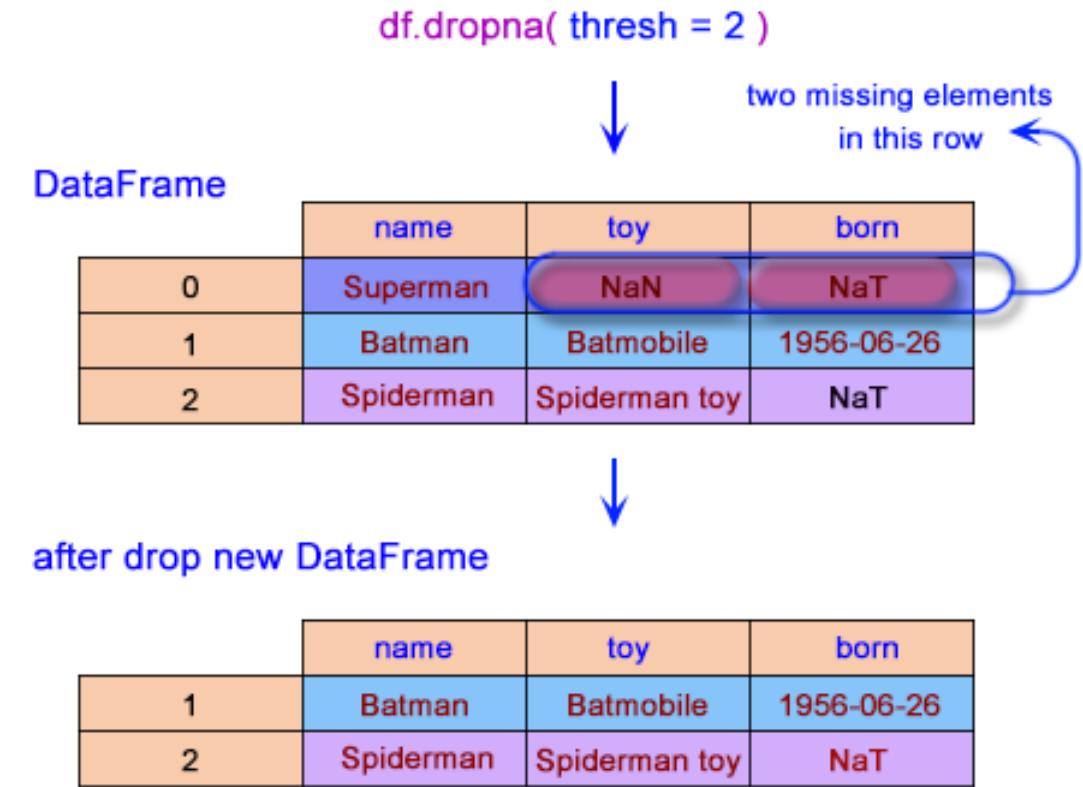
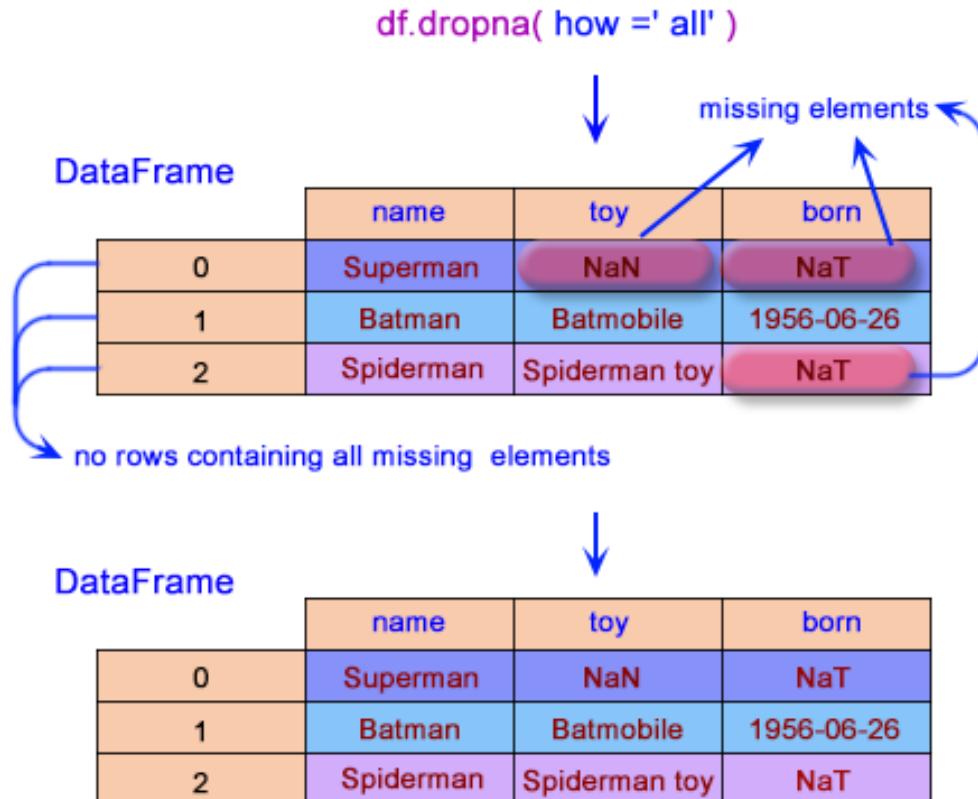
- **axis**: possible values are {0 or 'index', 1 or 'columns'}, default 0. If 0, drop rows with null values. If 1, drop columns with missing values.
- **how**: possible values are {'any', 'all'}, default 'any'. If 'any', drop the row/column if any of the values is null. If 'all', drop the row/column if all the values are missing.
- **thresh**: an int value to specify the threshold for the drop operation.
- **subset**: specifies the rows/columns to look for null values.
- **inplace**: a boolean value. If True, the source DataFrame is changed and None is returned.

# Cleaning Empty Cells

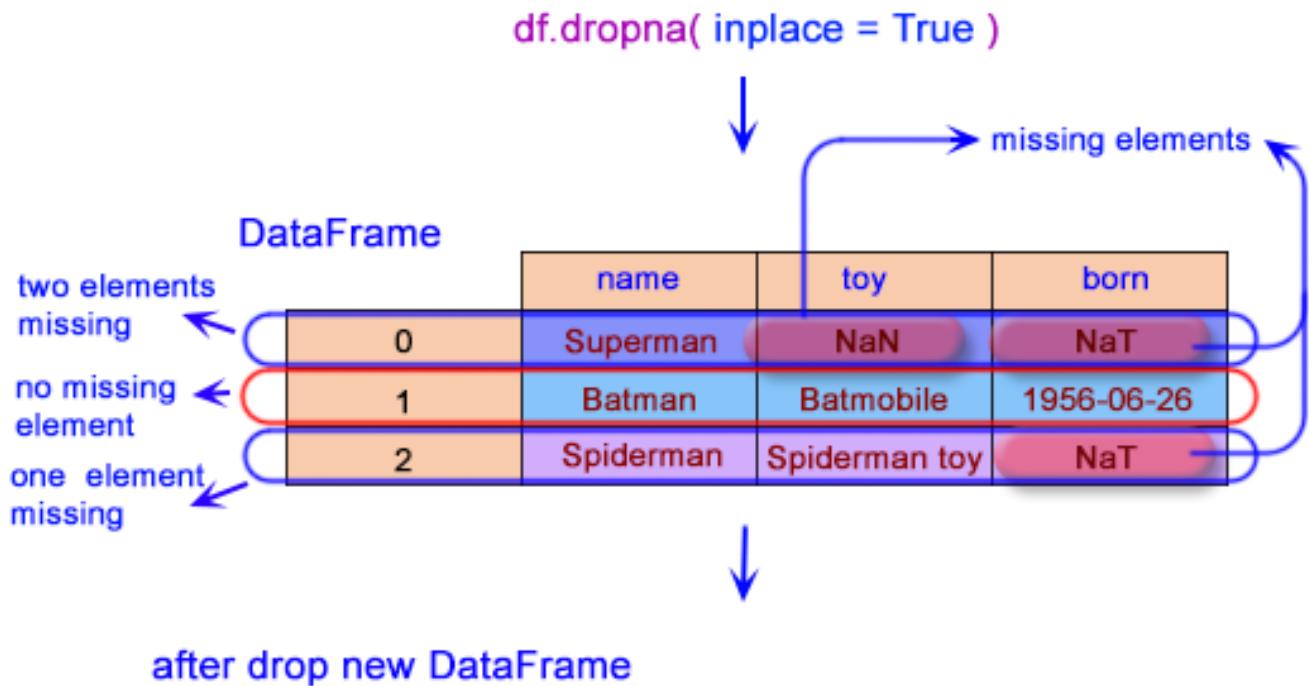
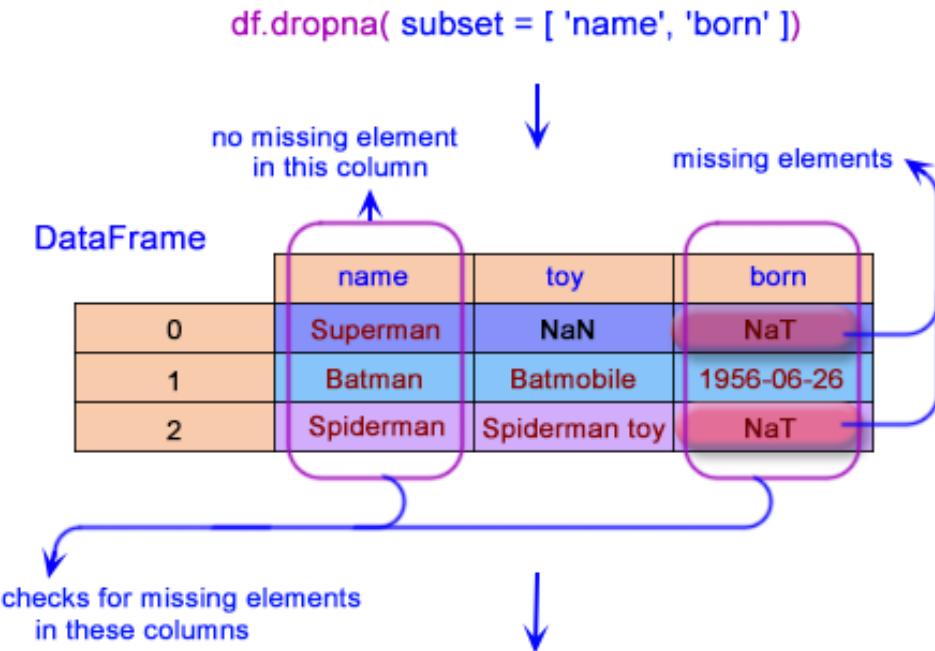


	name
0	Superman
1	Batman
2	Spiderman

# Cleaning Empty Cells



# Cleaning Empty Cells



# Fillna

The name of the  
DataFrame you  
want to operate on

The new value that  
will replace the  
missing values

```
myDataFrame.fillna(fill-value)
```

The name of  
the method



# Removing Duplicates

The name of your  
Pandas dataframe

```
your_dataframe.drop_duplicates(subset=, keep=)
```

The method  
name

Parameter specifying a  
subset of variables on which  
to look for duplicates

Parameter specifying  
which duplicate  
observation to keep

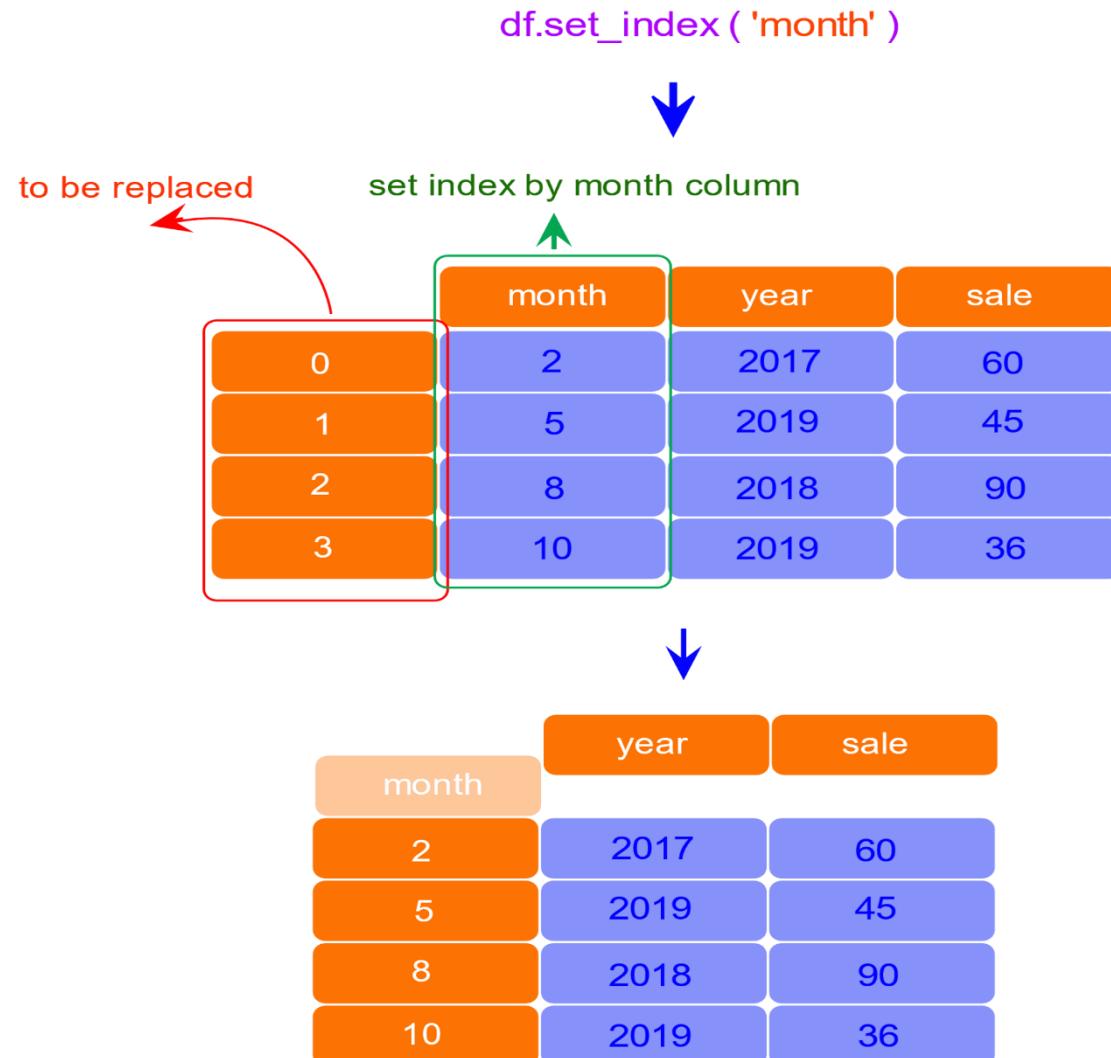
(i.e., 'first', 'last')

# Removing Duplicates

```
your_dataframe.drop_duplicates()
```

<b>name</b>	<b>region</b>	<b>sales</b>	<b>expense</b>
William	East	50000	42000
<del>William</del>	<del>East</del>	<del>50000</del>	<del>42000</del>
Emma	North	52000	43000
Emma	West	52000	43000
Anika	East	65000	44000
Anika	East	72000	53000

# Set DataFrame Index



# Reset DataFrame Index

The diagram illustrates the process of resetting a DataFrame index. On the left, a DataFrame with four rows and columns A, B, and C is shown. The index values are highlighted with a red border: 1, 3, 5, and 7. An arrow points to the right, where the same DataFrame is shown with its index values reset to 0, 1, 2, and 3, also highlighted with a green border.

	A	B	C
1	a1	1	c1
3	a2	2	c2
5	a3	3	c3
7	a4	4	c4

	A	B	C
0	a1	1	c1
1	a2	2	c2
2	a3	3	c3
3	a4	4	c4

The name of the DataFrame you want to operate on



```
myDataFrame.reset_index(level=, drop=, inplace= )
```

The name of the method



Which "level" of the index you want to remove, if the index has multiple levels (default is all levels)



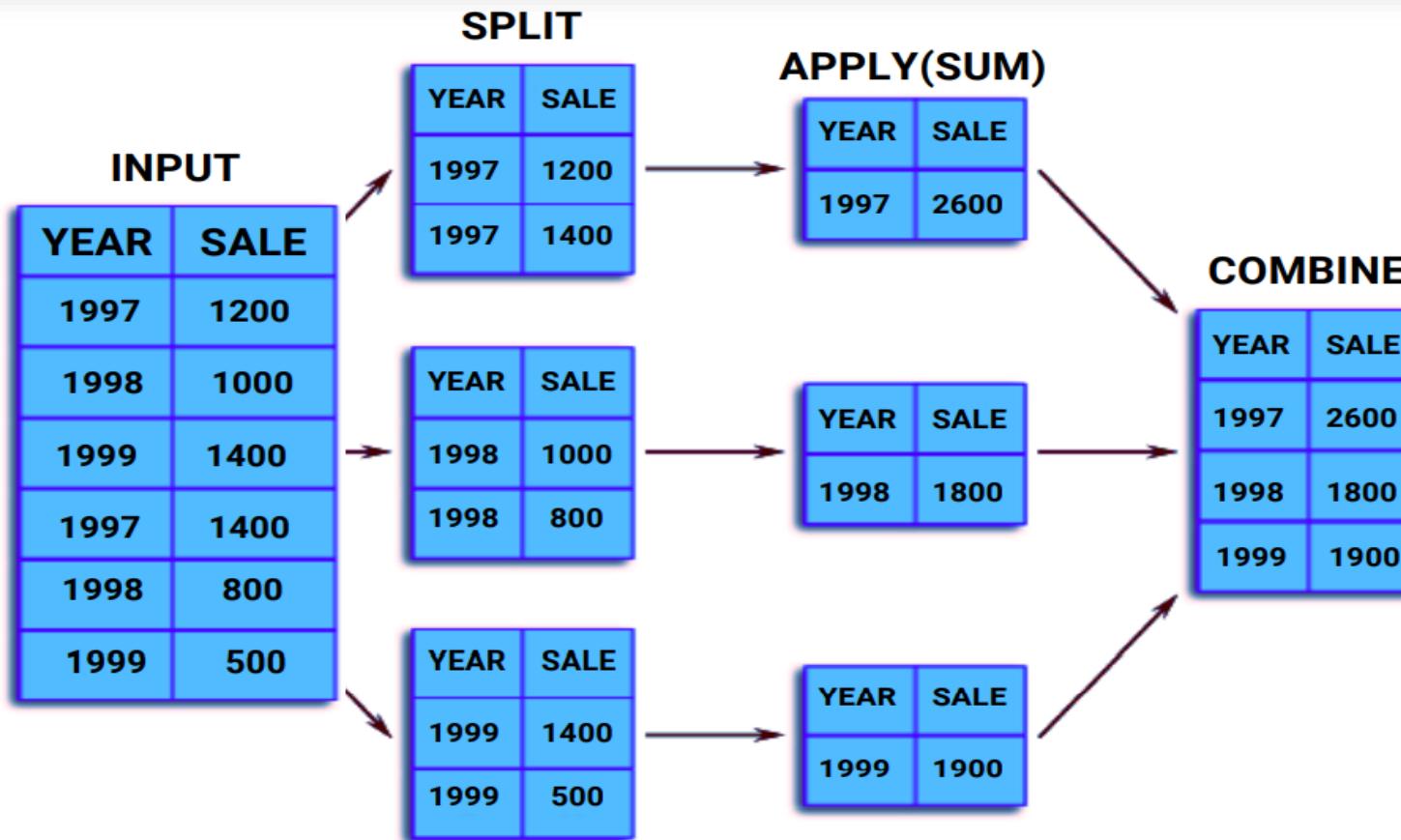
Whether you want to delete the index when it is removed (default is False)

Whether you want to operate on directly on the DataFrame (default is False)



# Group by

```
df.groupby('column_to_group')['column_to_agg'].agg_function()
```



# Group by

```
df.groupby('column_to_group')['column_to_agg'].agg_function()
```

```
df.groupby('Name')[AvgBill].sum()
```

Index	Name	Type	AvgBill
0	Lihoh Lihoh	Restaurant	\$45.32
1	Chambers	Restaurant	\$65.33
2	The Square	Bar	\$12.45
3	Tosca Cafe	Restaurant	\$180.34
4	Lihoh Lihoh	Restaurant	\$145.42
5	Chambers	Restaurant	\$25.35



**Lihoh Lihoh:** \$190.74  
**Chambers:** \$90.68  
**The Square:** \$12.45  
**Tosca Cafe:** \$180.34

# Aggregation Methods

Aggregation Method	Description
<code>.count()</code>	The number of non-null records
<code>.sum()</code>	The sum of the values
<code>.mean()</code>	The arithmetic mean of the values
<code>.median()</code>	The median of the values
<code>.min()</code>	The minimum value of the group
<code>.max()</code>	The maximum value of the group
<code>.mode()</code>	The most frequent value in the group
<code>.std()</code>	The standard deviation of the group
<code>.var()</code>	The variance of the group

# Group by Example

```
daily_spend_count = df.groupby('Day')['Debit'].count()  
daily_spend_sum = df.groupby('Day')['Debit'].sum()
```

1. Split the data by using  
values in the "Day" column

```
daily_spend = df.groupby('Day').agg({'Debit': ['sum', 'count']})
```

2. Perform both "sum" and "count"  
operations on the "Debit" column of  
the grouped data

```
df.groupby(['Category','Month'])['Debit'].sum()
```

# Sort

The name of the DataFrame you want to operate on

```
myDataFrame.sort_values(by, ascending =, inplace = )
```



The column or list of columns you want to sort by



Whether you want to operate on directly on the DataFrame (default is False)



The name of the method



Whether you want to sort in ascending order (default is True)



# Correlations

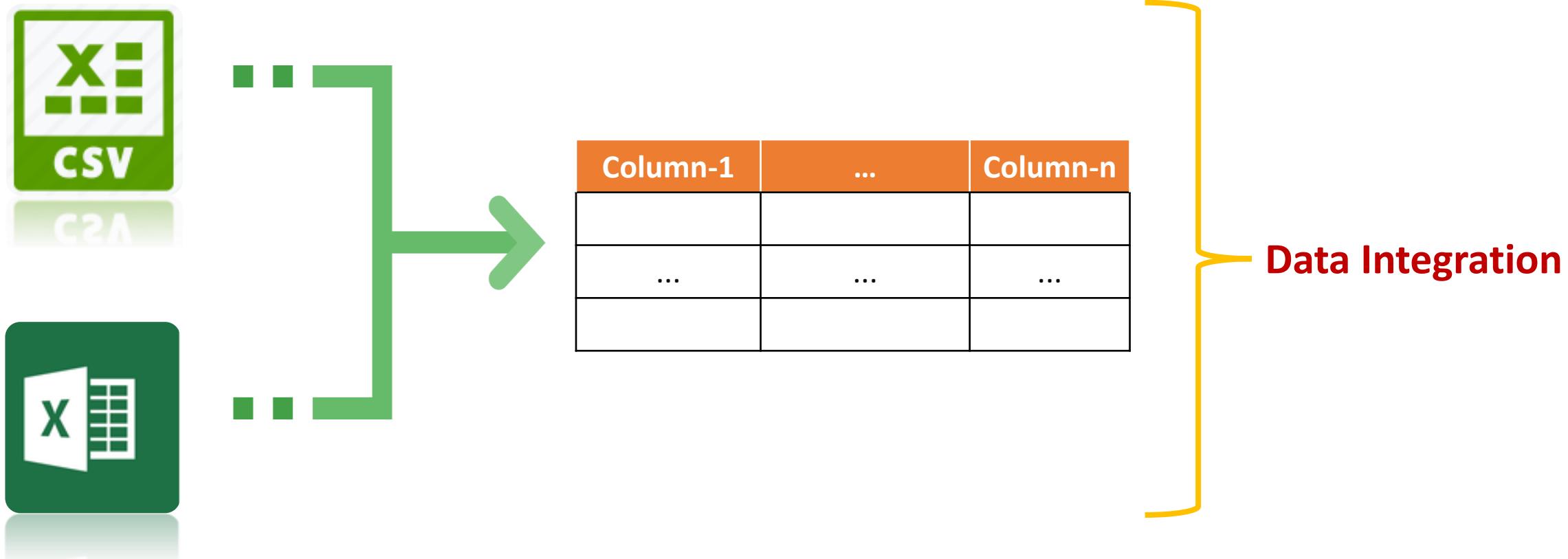
	Maths	Physics	History
0	78	81	53
1	85	77	65
2	67	63	95
3	69	74	87
4	53	46	63
5	81	72	58
6	93	88	73
7	74	76	42

df.corr()

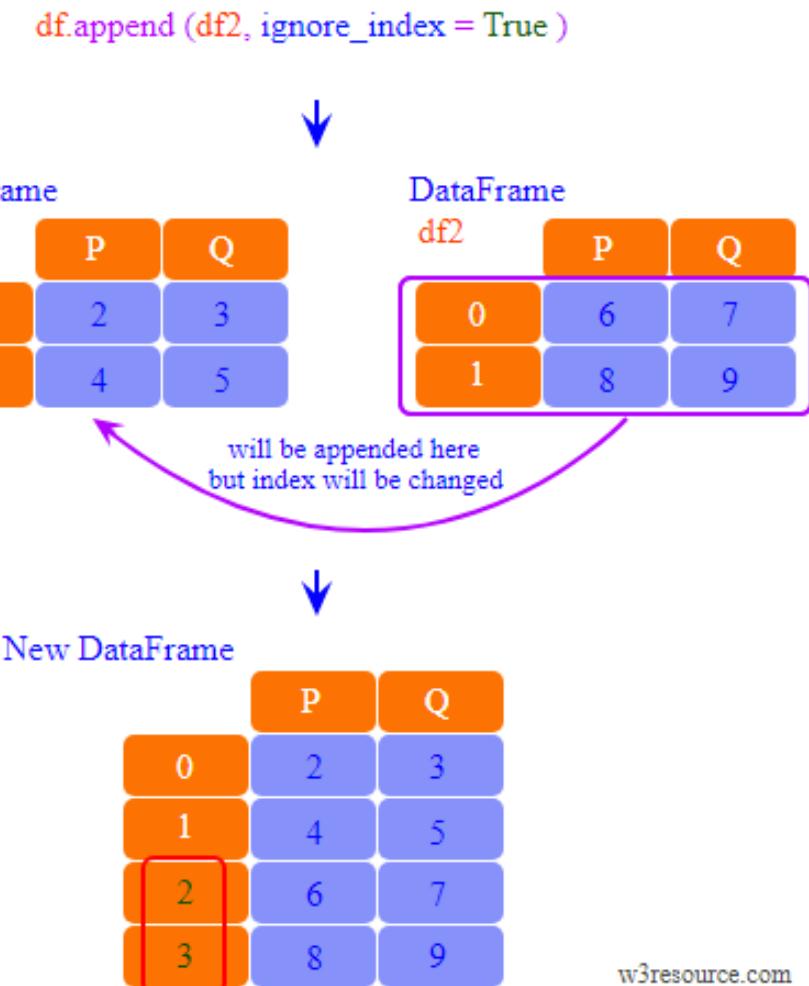
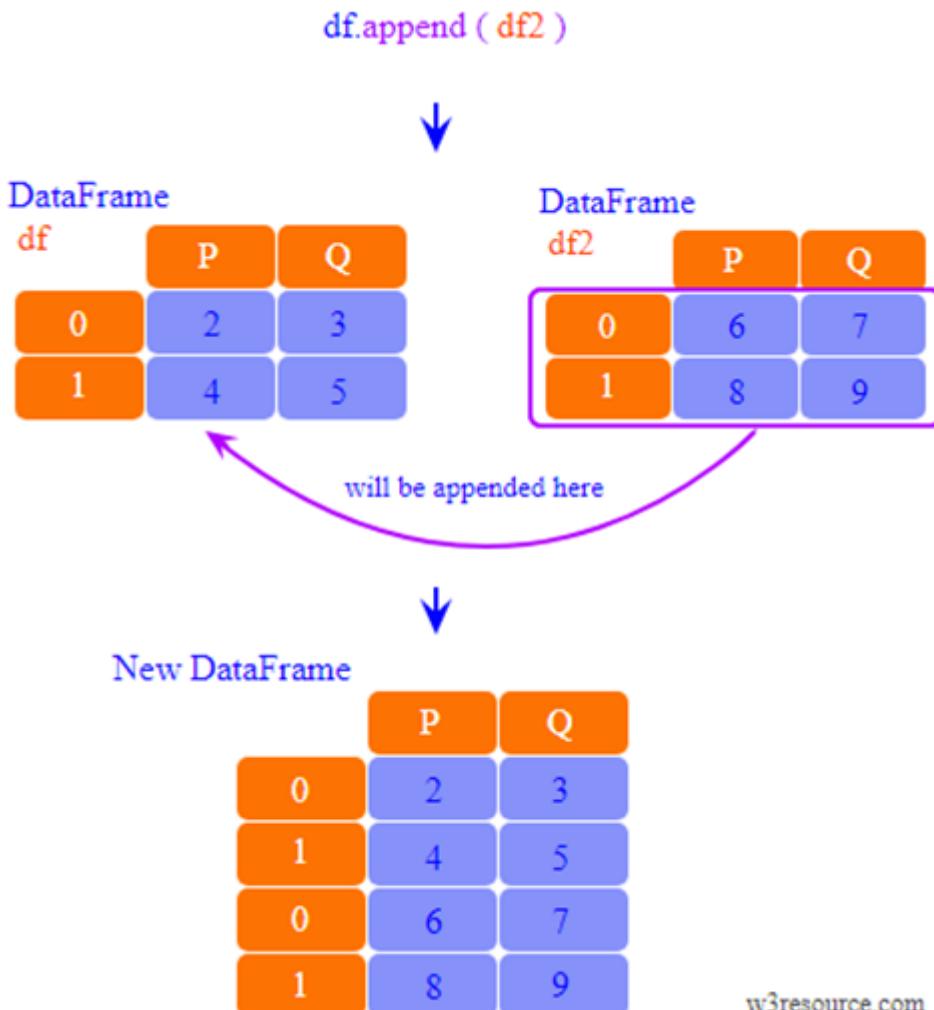


	Maths	Physics	History
Maths	1.000000	0.906340	-0.159063
Physics	0.906340	1.000000	-0.158783
History	-0.159063	-0.158783	1.000000

# Data Integration



# append



# Concatenate

	Country	Currency
0	Japan	Yen
1	China	chin
2	India	rupees

	capital	population
0	Jaya'pura	456750
1	Toronto	3456545
2	Aukland	89752

## Pandas concat( )

Default      axis = 0

axis = 1

	Country	Currency	capital	population
0	Japan	Yen	NaN	NaN
1	China	chin	NaN	NaN
2	India	rupees	NaN	NaN
0	NaN	NaN	Jaya'pura	456750.0
1	NaN	NaN	Toronto	3456545.0
2	NaN	NaN	Aukland	89752.0

```
1 pd.concat([data_3, my_df_drop], axis = 1)
```

	Country	Currency	capital	population
0	Japan	Yen	Jaya'pura	456750
1	China	chin	Toronto	3456545
2	India	rupees	Aukland	89752

# Concatenate

	orange	apple	grapes
0	3	0	7
1	2	3	14
2	0	7	6
3	1	2	15

	grapes	mango	banana	pear	pineapple
0	13	10	20	21	30
1	12	13	23	24	33
3	2	2	4	51	30
4	55	9	0	22	36
5	98	76	9	25	31

Concat with axis = 0  
is same as Append

Concat with axis = 1

Concat  
axis = 0  
Append  
Concat  
axis = 1

	orange	apple	grapes	mango	banana	pear	pineapple
0	3.0	0.0	7	NaN	NaN	NaN	NaN
1	2.0	3.0	14	NaN	NaN	NaN	NaN
2	0.0	7.0	6	NaN	NaN	NaN	NaN
3	1.0	2.0	15	NaN	NaN	NaN	NaN
0	NaN	NaN	13	10.0	20.0	21.0	30.0
1	NaN	NaN	12	13.0	23.0	24.0	33.0
3	NaN	NaN	2	2.0	4.0	51.0	30.0
4	NaN	NaN	55	9.0	0.0	22.0	36.0
5	NaN	NaN	98	76.0	9.0	25.0	31.0

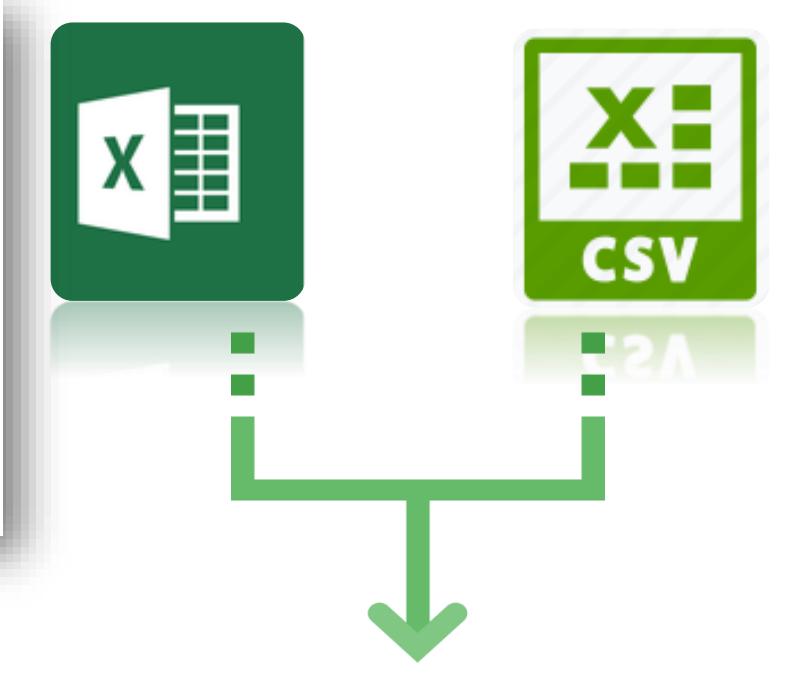
	orange	apple	grapes	grapes	mango	banana	pear	pineapple
0	3.0	0.0	7.0	13.0	10.0	20.0	21.0	30.0
1	2.0	3.0	14.0	12.0	13.0	23.0	24.0	33.0
2	0.0	7.0	6.0	NaN	NaN	NaN	NaN	NaN
3	1.0	2.0	15.0	2.0	2.0	4.0	51.0	30.0
4	NaN	NaN	NaN	55.0	9.0	0.0	22.0	36.0
5	NaN	NaN	NaN	98.0	76.0	9.0	25.0	31.0

# Merge Function

index	Year	Temperature
0	1956	16.99
1	1957	10.34
2	1958	21.01
3	1959	23.68
4	1960	24.59
5	1961	25.29
6	1962	8.77
7	1963	26.88
8	1964	15.04



index	Year	Rainfall
0	1956	1.01
1	1957	1.66
2	1958	3.5
3	1959	3.31
4	1960	3.61
5	1961	4.71
6	1962	2
7	1963	3.12
8	1964	1.96



A diagram illustrating the 'Merge Function'. It shows two separate data tables on the left and right, each with an icon above it (Excel for the first, CSV for the second). A large green arrow points downwards from both tables towards a third table at the bottom. This third table has columns labeled 'Year', 'Temperature', and 'Rainfall', with three rows of data below them, indicated by ellipses.

Year	Temperature	Rainfall
...	...	...

# Pandas Merge

`df.merge(right=other_df, on='common_column', how='how_to_join')`

df



index	Year	Rainfall
0	1956	1.01
1	1957	1.66
2	1958	3.5
3	1959	3.31
4	1960	3.61
5	1961	4.71

+

Other\_df



index	Year	Temperature
0	1956	16.99
1	1957	10.34
2	1958	21.01
3	1959	23.68
4	1960	24.59
5	1961	25.29

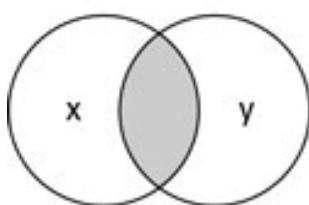
=

index	Year	Temperature
0	1956	16.99
1	1957	10.34
2	1958	21.01
3	1959	23.68
4	1960	24.59
5	1961	25.29



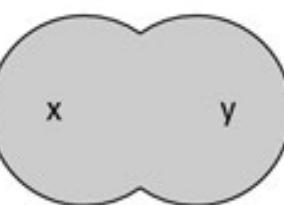
# Pandas Merge

**how='inner'**



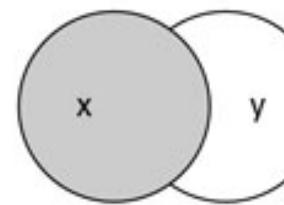
natural join

**how='outer'**



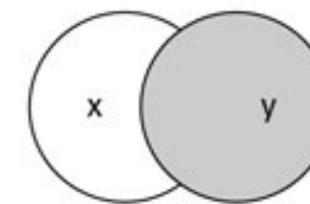
full outer join

**how='left'**



left outer join

**how='right'**



right outer join

index	Year	Temperature
0	1956	16.99
1	1957	10.34
2	1958	21.01
3	1959	23.68
4	1960	24.59
5	1961	25.29

index	Year	Rainfall
0	1956	1.01
1	1958	3.5
2	1959	3.31
3	1960	3.61
4	1962	2
5	1963	3.12

index	Year	Temperature	Rainfall
0	1956	16.99	1.01
1	1957	10.34	Nan
2	1958	21.01	3.5
3	1959	23.68	3.31
4	1960	24.59	3.61
5	1961	25.29	Nan
6	1962	Nan	2
7	1963	Nan	3.12

# Join

`df_left.join(df_right, how='left')`

df_names	
	Name
Symbol	
APPL	Apple
MSFT	Mircosoft
TSLA	Tesla
GOOG	Google
NFLX	Netflix

df_portfolio	
	Shares
Symbol	
TSLA	20
APPL	50
GOOG	50
AMZN	100

`df_names.join(df_portfolio, how='left')`

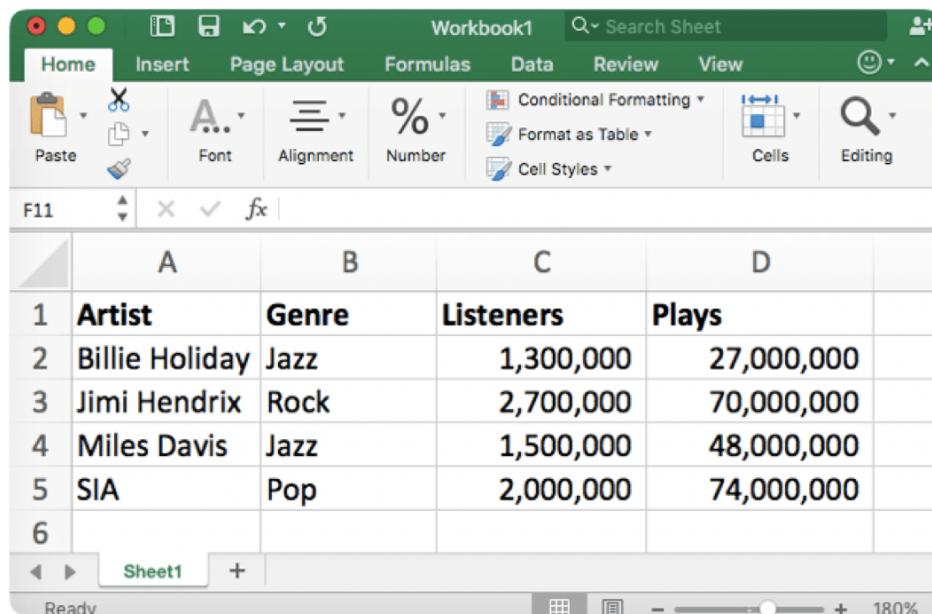
	Name	Shares
APPL	Apple	50.0
MSFT	Mircosoft	NaN
TSLA	Tesla	20.0
GOOG	Google	50.0
NFLX	Netflix	NaN

# Pandas concat Vs append Vs join Vs merge

- **Concat** gives the flexibility to join based on the axis( all **rows** or **all columns**)
- **Append** is the specific case(`axis=0, join='outer'`) of **concat**
- **Merge** is based on any particular **column** each of the two dataframes, this columns are variables on like '`left_on`', '`right_on`', '`on`'.
- **Join** is based on the indexes (set by **set\_index**) on how variable =['`left`', '`right`', '`inner`', '`outer`']

# Reading files

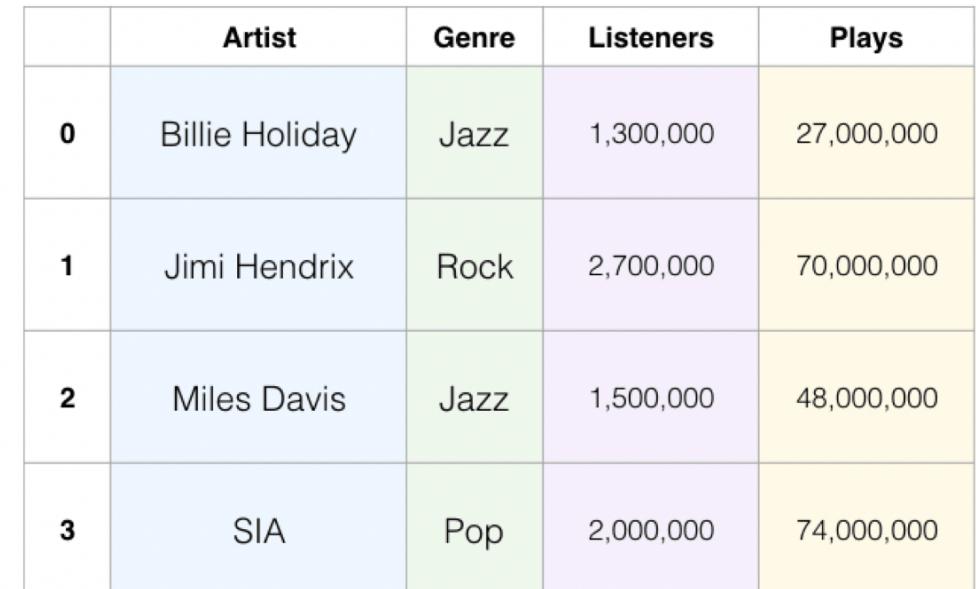
music.csv



A screenshot of Microsoft Excel showing a CSV file named "music.csv". The spreadsheet has four columns: Artist, Genre, Listeners, and Plays. The data is as follows:

	Artist	Genre	Listeners	Plays
1	Billie Holiday	Jazz	1,300,000	27,000,000
2	Jimi Hendrix	Rock	2,700,000	70,000,000
3	Miles Davis	Jazz	1,500,000	48,000,000
4	SIA	Pop	2,000,000	74,000,000

pandas.read\_csv('music.csv')



A screenshot of a Jupyter Notebook cell showing the command `pandas.read_csv('music.csv')` and its resulting output. The output is a DataFrame with the following structure:

	Artist	Genre	Listeners	Plays
0	Billie Holiday	Jazz	1,300,000	27,000,000
1	Jimi Hendrix	Rock	2,700,000	70,000,000
2	Miles Davis	Jazz	1,500,000	48,000,000
3	SIA	Pop	2,000,000	74,000,000

# Export DataFrame to CSV File

*Export Pandas DataFrame to CSV File*

DataFrame

name

```
df.to_csv(r'Path to save CSV file\File Name.csv')
```

Use .txt to  
save as text file

Path e.g.

D:\Python\Tutorial\Example1.csv

File name e.g.



Key to DataScience

# Replace

The name of the dataframe  
you want to operate on



The a list of old values  
(passed to the `to_replace` parameter)



```
your_dataframe.replace(to_replace= [...], value= new_val)
```



The method name



The new value  
(passed to the `value` parameter)

```
sales_data.replace(to_replace = -1, value = np.NaN)
```

```
sales_data.replace(to_replace = [-1,-999], value = np.NaN)
```

# Replace

The name of the dataframe  
you want to operate on



```
your_dataframe.replace({ 'col_name': { 'old_val': 'new_val' } })
```

The name of the column  
you want to operate on



The name of the new  
replacement value



The method name



The old value to  
replace



# dtypes

df.dtypes



DataFrame	float	int	datetime	string
0	2.0	2	2019-02-10	'f1'



columns appears in series

float	float64
int	int64
datetime	datetime64[ns]
string	object

# astype

The name of your  
Pandas Series

The Python datatype  
that you want to use  
(e.g., category, int8, float16, etc)

```
your_series.astype(datatype)
```

The name of  
the method

# astype

```
your_dataframe.astype( { 'column': 'datatype' } )
```

The name of  
the dataframe  
column

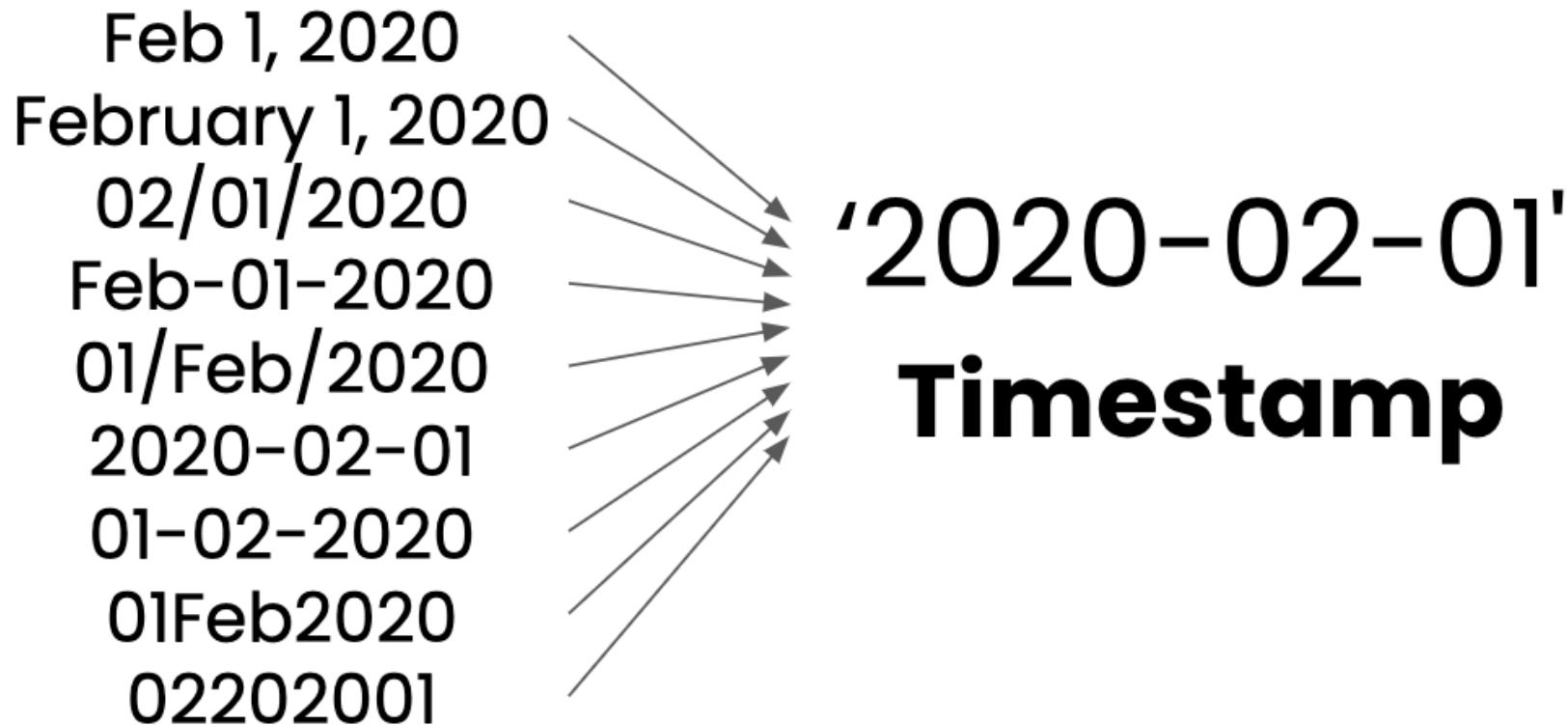


The datatype you want  
to use for that column  
(e.g., category, int8, float16, etc)

# to\_datetime()

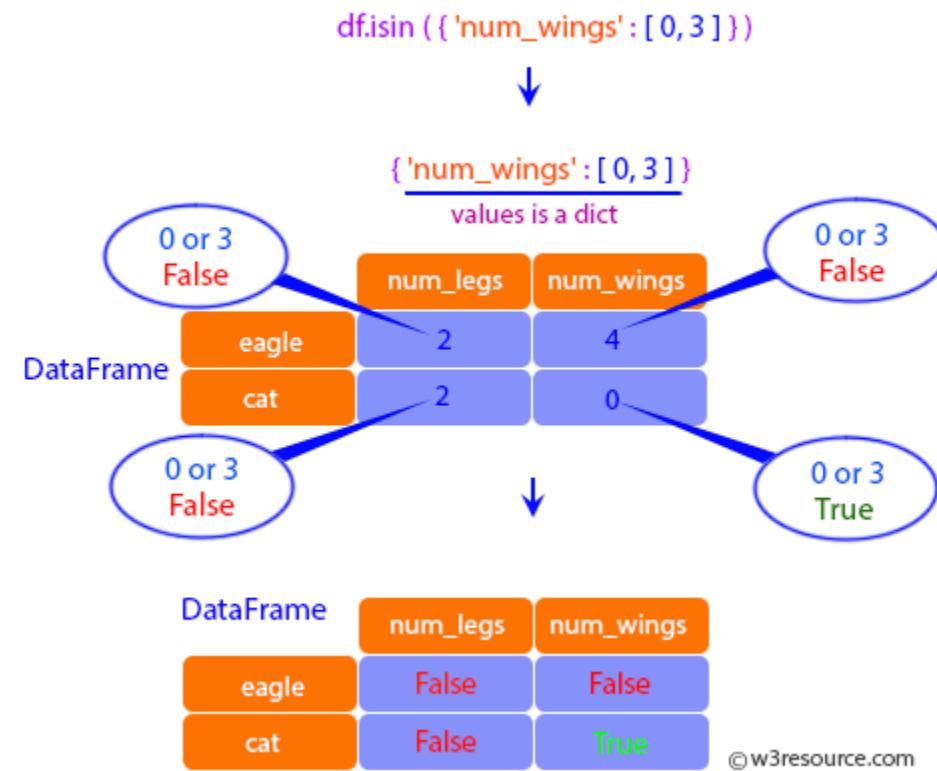
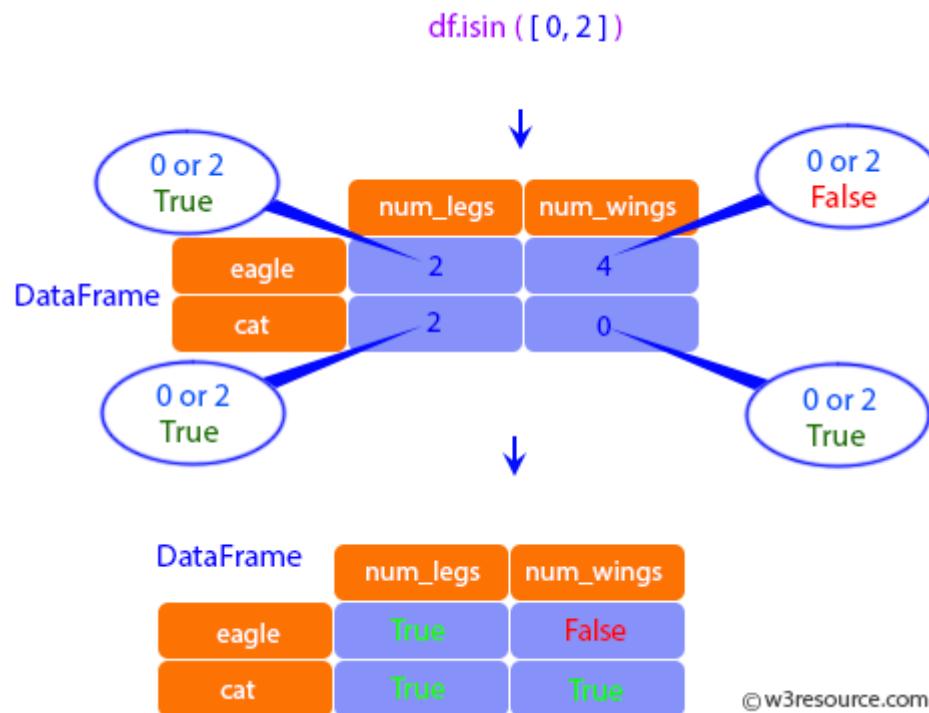
`pd.to_datetime(format='Your_Datetime_Format')`

"Given a format, convert a string to a datetime object"



```
earthquakes['date_parsed']=pd.to_datetime(earthquakes['Date'],format="%m/%d/%y")
```

# isin



# where

```
pd.DataFrame.where(cond=df<90, other="A+")
```

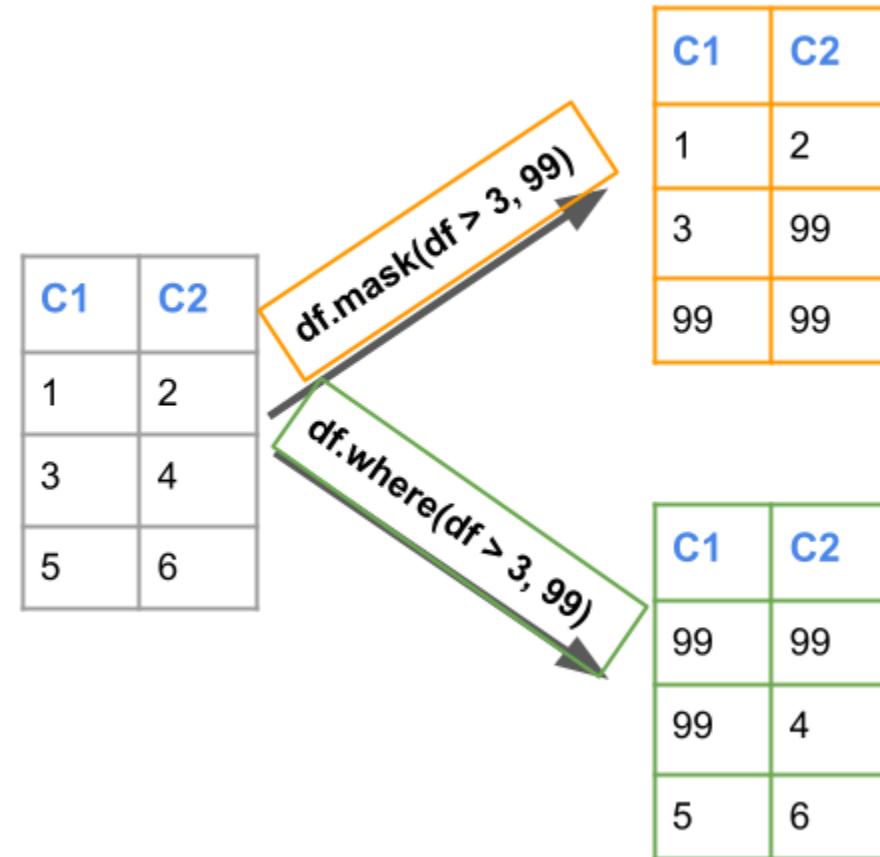
“Where a condition is false, replace a value”

Index	Test1	Test2	Test3
Bob	51	92	14
Sally	71	60	20
Frank	82	86	74
Patty	74	87	99

Index	Test1	Test2	Test3
Bob	51	A+	14
Sally	71	60	20
Frank	82	86	74
Patty	74	87	A+

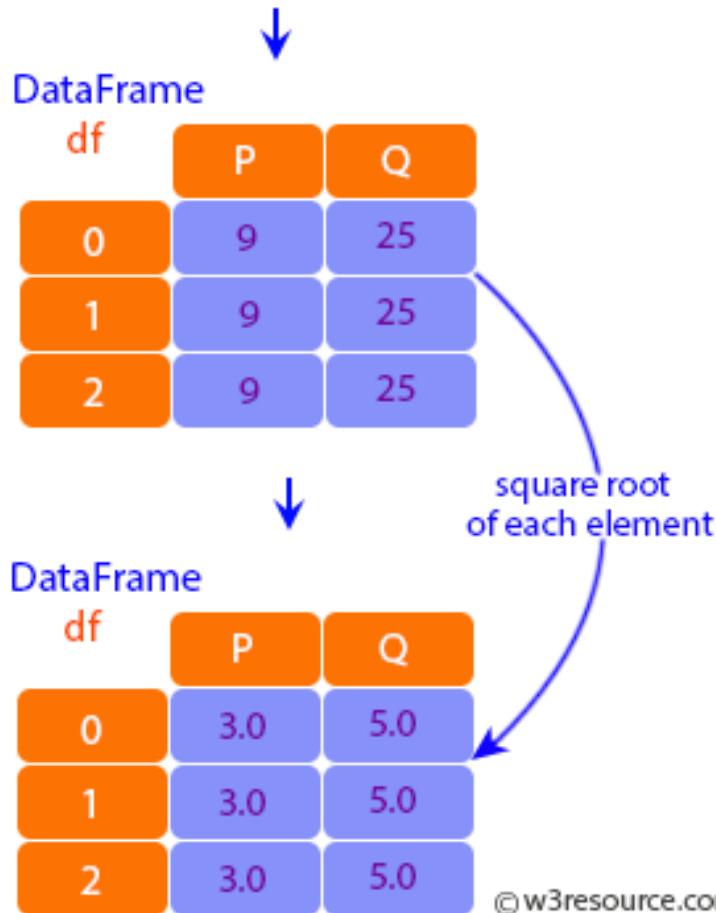
# mask

## Pandas mask() vs where()



# Apply Function

`df.apply( np.sqrt )`



# Apply Function

`df.apply ( np.sum, axis = 0 )`

↓

df	P	Q
0	9	25
1	9	25
2	9	25

`np.sum`  
function apply  
for each column

↓  
`axis = 0`  
`axis 0 represents index`

for column P = index 0 + index 1 + index 2 =  $9 + 9 + 9 = 27$

for column Q = index 0 + index 1 + index 2 =  $25 + 25 + 25 = 75$

↓  
P    27

Q    75

©w3resource.com

`df.apply ( np.sum, axis = 1 )`

↓

df	P	Q
0	9	25
1	9	25
2	9	25

`np.sum`  
function apply  
for each row or index

↓  
`axis = 1`  
`axis 1 represents column`

for index 0 = column P + column Q =  $9 + 25 = 34$

for index 1 = column P + column Q =  $9 + 25 = 34$

for index 2 = column P + column Q =  $9 + 25 = 34$

↓  
0    34  
1    34  
2    34

©w3resource.com

# unique & nunique

Age	Department
26	Sales
28	Sales
27	Accounting
32	HR

`df.unique()`



Age	4
Department	3

Distinct Values in Each Column

Height	Weight	Team
167	65	A
175	70	A
170	72	B
186	80	B
190	86	B
188	94	C
158	50	A
169	58	C
183	78	B
180	85	C

`df['Team'].unique()`



A
B
C

Unique Values in a Column

# value\_counts

region
East
North
East
South
West
West

.value\_counts (...)



value	count
East	2
North	1
South	1
West	2

# select\_dtypes

`df.select_dtypes(include, exclude)`

```
1. # select all numeric columns  
2. df.select_dtypes(include='number')
```

Output:

	<b>Age</b>	<b>Salary</b>
<b>0</b>	26	70000.0
<b>1</b>	28	80000.0
<b>2</b>	27	72000.5
<b>3</b>	32	66000.0

# interpolate

	Price
2020-12-04	10.0
2020-12-05	20.0
2020-12-06	NaN
2020-12-07	30.0
2020-12-08	50.0
2020-12-09	20.0
2020-12-10	NaN
2020-12-11	100.0
2020-12-12	30.0
2020-12-13	NaN

```
df.interpolate()
```

	Price
2020-12-04	10.0
2020-12-05	20.0
2020-12-06	25.0
2020-12-07	30.0
2020-12-08	50.0
2020-12-09	20.0
2020-12-10	60.0
2020-12-11	100.0
2020-12-12	30.0
2020-12-13	30.0

# get\_dummies

The name of  
the function

The column or columns you want  
to convert to dummy variables

(applicable for dataframes)

```
pd.get_dummies(data_object, columns=, drop_first=)
```

The object you  
want to operate on  
(e.g., a dataframe, a Series, etc)

A True/False indicator  
that specifies if you  
want to drop the first  
level of the categorical  
variable



# melt

Country	1800	1801	1802	1803	1804	1805	1806	1807	1808	1809	1810	1811	1812	1813	1814
Afghanistan	3.28M														
Angola	1.57M														
Albania	400k	402k	404k	405k	407k	409k	411k	413k	414k	416k	418k	420k	422k	424k	426k

```
df=df.melt(id_vars=['country'],var_name='Year', value_name='total_Population')
```

•**id\_vars** : *tuple, list, or ndarray, optional*

Column(s) to use as identifier variables.

•**value\_vars** : *tuple, list, or ndarray, optional*

Column(s) to unpivot/melt. If not specified, uses all columns that are not set as id\_vars.

•**var\_name** : *scalar*

Name to use for the ‘variable’ column. If None it uses frame.columns.name or ‘variable’.

•**value\_name** : *scalar, default ‘value’*

Name to use for the ‘value’ column.

```
1 df[df['country']=='Afghanistan']  
✓ 0.4s
```

	country	Year	total_Population
0	Afghanistan	1800	3.28M
197	Afghanistan	1801	3.28M
394	Afghanistan	1802	3.28M
591	Afghanistan	1803	3.28M
788	Afghanistan	1804	3.28M
...	...	...	...
58312	Afghanistan	2096	75.8M
58509	Afghanistan	2097	75.6M
58706	Afghanistan	2098	75.4M
58903	Afghanistan	2099	75.2M
59100	Afghanistan	2100	74.9M

# Reshaping Data in Pandas

## Melt

df3

	first	last	height	weight
0	John	Doe	5.5	130
1	Mary	Bo	6.0	150

df3.melt(id\_vars=['first', 'last'])



	first	last	variable	value
0	John	Doe	height	5.5
1	Mary	Bo	height	6.0
2	John	Doe	weight	130
3	Mary	Bo	weight	150

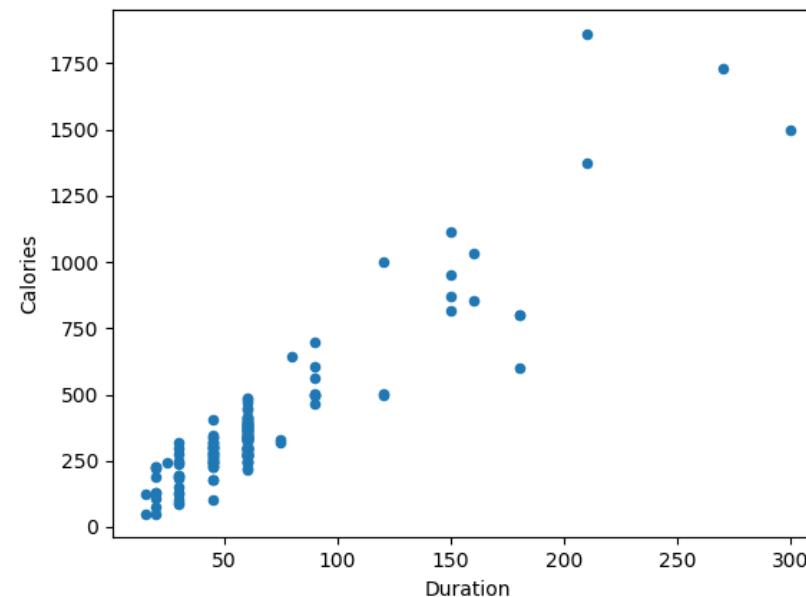
# Pandas

- duplicated
- Isna = Isnull
- To\_numpy
- Columns
- Index

# Plot

- [pandas.DataFrame.plot.area](#)
- [pandas.DataFrame.plot.bar](#)
- [pandas.DataFrame.plot.barch](#)
- [pandas.DataFrame.plot.box](#)
- [pandas.DataFrame.plot.density](#)
- [pandas.DataFrame.plot.hexbin](#)
- [pandas.DataFrame.plot.hist](#)
- [pandas.DataFrame.plot.kde](#)
- [pandas.DataFrame.plot.line](#)
- [pandas.DataFrame.plot.pie](#)
- [pandas.DataFrame.plot.scatter](#)
- [pandas.DataFrame.boxplot](#)

```
df.plot(kind = 'scatter', x = 'Duration', y = 'Calories')  
plt.show()
```



# Practical Lab

- Inconsistent Data Entry (EX1)
- Parsing Dates (EX2)
- Character Encodings (EX3)
- Categorical Data (EX4)
- Handle Missing Values (EX5)
- Handle Outliers (EX6)
- Handle Duplicates(EX7)