Data Mining and Accounting Analytics -Python Basics

Dr. Yi Long (Neal)

Outline

- Introduction to Python
- Python Basics(syntax)
- Control flow and basic data structure
- Data Understanding
- Pandas and Numpy

Control Flow in Python

- **Conditional statements :** if … else …
- **Loop statements:** for … / while …
- **Function call:** def function:

Conditional Statements (1)

- **`if` statement**: Executes a group of statements only if a certain condition is true. Otherwise, the statements are skipped.
  - ✓ Syntax:

```
if condition:
    statements # executed if true condition
```

- **`if/else` statement**: Executes one block of statements if a certain condition is True, and a second block of statements if it is FalseSyntax:
  - ✓ Syntax:

```
if condition:
    statements # executed if true condition
else:
    statements # executed if false condition
```

Conditional Statements (2)

- **`if/else` statement can be chained to a big if** : Conditions are tested in the order they appear and the corresponding block of python statements are executed, **but not the rest**.
  - ✓ Syntax:

```
if condition1:
    some python statements # executed if true condition1
elif condition2:
    some python statements # executed if true condition2
elif condition3:
    some python statements # executed if true condition3
else:
    statements # executed if no conditions are true
```

Conditional Statements (3)

- **Example code: get the absolute value**
  - ✓ Syntax:

```
if x < 0:
    print "x is negative“
    x = -1*x
elif x >0:
    print "x is positive "
else:
    print "x is 0“
print x
```

Loop Statements – while loops

- **`while loop:`** repeats the statements **as long as** the condition is true.
  - ✓ Syntax:

```
while condition:
    statements
```

  - ✓ Example:

```
count = 0
while x > 0:
    x = x // 2          # truncating division
    count += 1
print ("The approximate log2 of x  is", count)
```

Loop Statements – for loops

- **`for loop:`** repeats a set of statements over a group of values.
  - ✓ Syntax:

    for *variableName* ***in*** *groupOfValues:*

    statements for manipulating variableName
- The range function : **range(*start*, *stop*, *step*): r**eturn the integers between **start (inclusive, =0 by default**) and **stop** (**exclusive**) by **step (=1 by default)**
  - ✓ range(5) = range(0,5,1), return 0,1,2,3,4 sequentially
  - ✓ range(4,-1,-1), return 4,3,2,1,0 sequentially
- range function is usually used with for loop
  - ✓ for x in range(5):

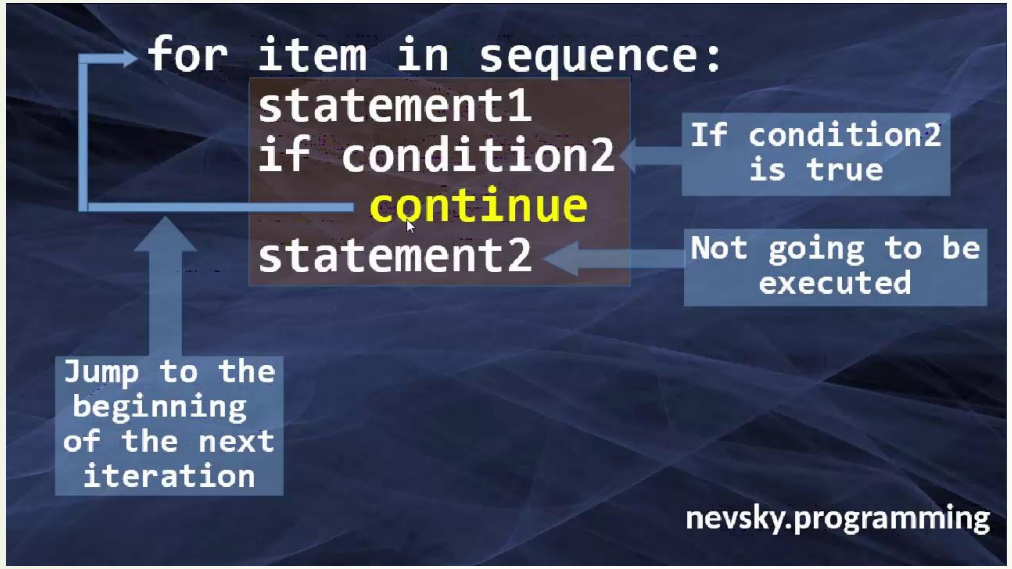
    print (x)

    ⟺

    print 1
    print 2
    print 3
    print 4
    print 5

continue Statements in Loops

- **`continue`:** continues with the next iteration of the loop



for student in cuhk_applicants:

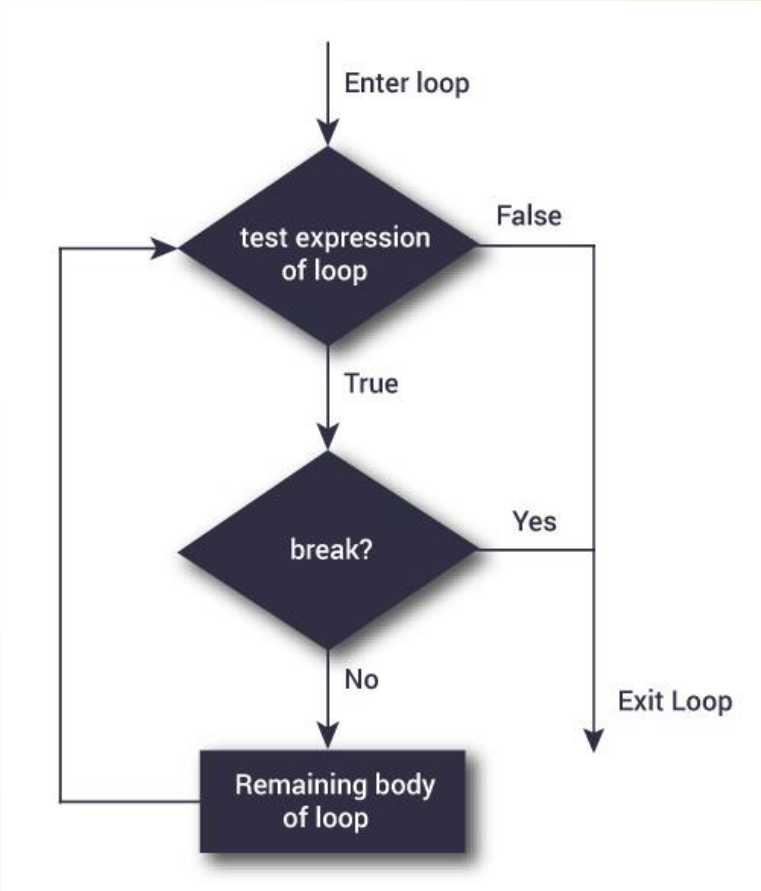
```
    if student.gpa < 80:
        continue
    check_other_things(student)
```

break Statements in Loops

- **break** will terminate the innermost loop.

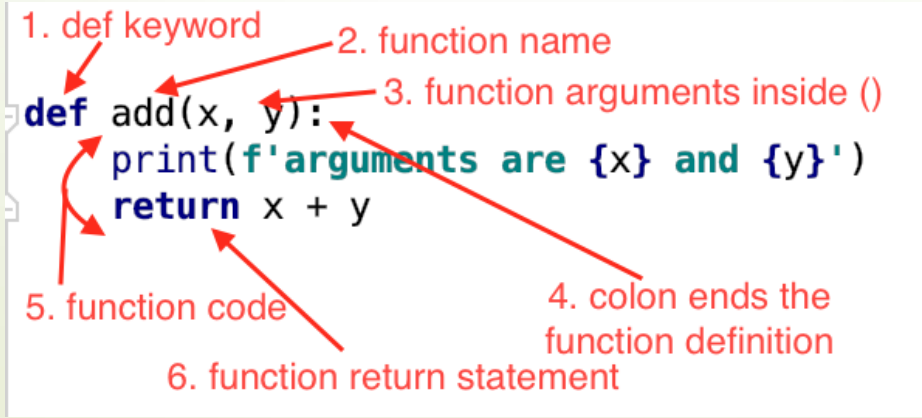
```
i=0
for val in "string":
    if val == "i":
        break
    i+=1
    print(i)
print("The end")
```

“string”.find(i)



Python Functions(1)

- There are two types of functions in Python
  - ✓ Built-in functions: type(), max(), input(), range() …
  - ✓ User-defined functions by packages or us
- We can call a function by " function_name (arg1, arg2, arg3…)"
- Functions can be defined as follows:
  - ✓ In Python a function is some reusable code that takes arguments(s) as input does some computation and then optionally returns a result or results
  - ✓ A function can be defined with **def** as follows



Python Functions (2)

- Can define defaults for arguments that need not be passed
  - ✓ def function_name(arg1, arg2, arg3=4):
    statement
  - ✓ Call function with arguments
    - By position
    - By keyword

```python
#coding=utf8
"""
Created on Thu Sep 01 18:04:16 2018

@author: Neal LONG
"""

def calc_perimeter(height,width=10):
    return 2*(height+width)


if __name__=="__main__":
    #get and convert the input height of rectangular
    x=float(input('Please input height of rectangular:'))
    #get and convert the input width of rectangular
    y=float(input('Please input width of rectangular:'))

    #calculate the perimeter of rectangular
    print(calc_perimeter(x,y))

    #calculate the perimeter of rectangular with default width
    print(calc_perimeter(x))

```

Basic Data Structure in Python

- "**Data structure** is a particular way of organizing data in a computer so that it can be used efficiently."-- Wikipedia
  - ✓ Sequences (list, tuple): **ordered/indexed** sequences of objects
  - ✓ Set: collections of unique but unordered objects
  - ✓ Dictionary: Store ***pairs*** of (key, value) which indexed by key

- They share following functions:
  - ✓ len(X): return the number of objects in data structure X
  - ✓ for x in X: iterate the object in data structure X one by one
  - ✓ sorted(X,reverse=False): return Return a **new list** containing all items from X in ascending(if reverse=True) order.
  - ✓ x in X return True if data structure X contains element x

Sequences

- **List**: list_a = [1,2,3]
  - ✓ Mutable, can add, delete, replace, reorder stored objects
  - ✓ Defined using square brackets (and commas)
  - ✓ Have useful functions to update: append(), extend() ,del(), pop() ….
- **Tuple:** tuple_b=(1,2,3)
  - ✓ Immutable, cannot be updated, including add, delete or reorder
  - ✓ Defined using square parentheses (and commas)
  - ✓ A tuple with a single element must have a comma inside the parentheses: tuple_c=(1,)
- **String**: Conceptually very much like a tuple
  - ✓ Immutable, cannot be updated, including add, delete or reorder

Sequences (tuple, list, string) Slicing

- **Slice sequence will return a subsequence as a new list (sequence can be list or tuple or String)**
  - ✓ index numbered from 0 to len(seq) -1
  - ✓ a[start:end:step] # items with index from **start** through **end-1** by **step**
  - ✓ a[start:] # items start through the rest of the array
  - ✓ a[:end] # items from the beginning through end-1
  - ✓ a[:] # a copy of the whole array
  - ✓ Negative number means index count from the end

Positive indexes	0	1	2	3	4	5	6	7	8	9	10	11
String	P	Y	T	H	O	N		R	O	C	K	S
Negative indexes	–12	–11	–10	–9	–8	–7	–6	–5	–4	–3	–2	–1

X[3:5]='HO' if X is String (or [H','O'] if X is list/tuple)
X[7:] = "ROCKS"
X[11:7:-1] = "SKCO"
X[:6] = "PYTHON"
X[1:5:2] = "YH"
X[-9:--6] = "HON

Set

- **Set**: set_a= {1,2,3} or set_a = set([1,2,3])
  - ✓ Mutable collection of unique unordered immutable objects
  - ✓ Set cannot be indexed (unordered)
  - ✓ Empty set can only be defined as empty_set = set()
  - ✓ Sets work like the mathematical concept of sets
  - ✓ Have useful functions to update: update(), remove() ,add(), discard() ….
  - ✓ Stored objects must be immutable type: int, string, float, tuple …
  - ✓ set[(1,2,2,3,4,3)] =set ([1,2,3,4]) =set([2,3,4,1])
  - ✓ {(1,2),(3,4),(1,2),(2,3)} = {(1,2),(2,3),(3,4)}
  - ✓ {[1,2],[3,4],[1,2],[2,3]} is error

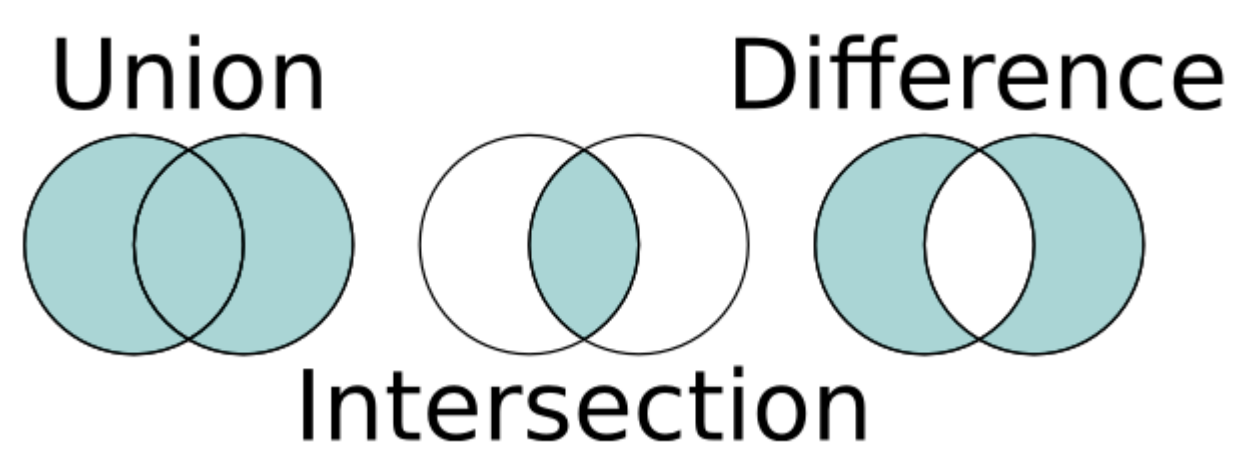
```
In [2]: {[1,2],[3,4],[1,2],[2,3]}
Traceback (most recent call last):

  File "<ipython-input-2-e7a3ea9195a1>", line 1, in <module>
    {[1,2],[3,4],[1,2],[2,3]}

TypeError: unhashable type: 'list'
```

Set Operators

- Two Set: set_a= {1,2,3} , set_b = {2,3,4}, set_c = {1,2}
  - ✓ Difference: set_a- set_b={1}, set_b- set_a={4}
  - ✓ Join(intersection): set_a & set_b={2,3}
  - ✓ Union: set_a | set_b={1,2,3,4}
  - ✓ Subset: set_c in set_a= True, set_c in set_b= False



Dictionary

- **Dict**: A mutable collection of unique unordered immutable keys paired with a mutable object.
  - ✓ dict_a=dict(), dict_a={}, dict_c={key1:value1, key2:value2 …},
  - ✓ Dictionary: fast lookup table for explanation(value) of word(key)
  - ✓ uni_loc={'CUHK': 'HK', 'PKU':'Peking', 'NUS', 'SG'}
  - ✓ uni_loc['CUHK'] will quickly give you 'HK'
  - ✓ Key must be immutable type: int, string, float, tuple …
  - ✓ Value can be any objects

Dictionary Operators

- **Dict**: assignment can add or update values
  - ✓ uni_loc['MIT'] = 'NYC'(add), uni_loc['MIT'] = 'Boston'(update)
  - ✓ uni_loc['MIT'] finally stores ''Boston'
  - ✓ for x in uni_loc: will iterate keys, i.e., university here
  - ✓ Similarly, x in uni_loc will just search x in keys of uni_loc, 'Boston' in uni_loc return False
  - ✓ for uni, loc in uni_loc.items(): will iterate (key,value) pairs
  - ✓ uni_loc['Boston'] will generate a KeyError

Mutable Vs. Immutable

- **Mutable data types: stored values can be changed in place (memory)**
  - ✓ Such as : list, set, dict
  - ✓ Can be changed via functions like, append, add, del, remove

- **Immutable: stored values can not be changed in place (memory)**
  - ✓ Such as int, float, string, tuple
  - ✓ Cannot be changed, or the memory address will be changed as well

```
In [6]: b=[1,2,3]

In [7]: id(b)
Out[7]: 169128520

In [8]: b.append(4)

In [9]: b
Out[9]: [1, 2, 3, 4]

In [10]: id(b)
Out[10]: 169128520
```

```
In [1]: a=10

In [2]: id(a)
Out[2]: 503311088

In [3]: a+=2

In [4]: a
Out[4]: 12

In [5]: id(a)
Out[5]: 503311152
```

Performance Tips

- A look-up in a **set** or a **dict** is very fast – **constant** time. This means that if does not matter how big the set/dict is, the look-up takes the same amount of time. In contrast, the look-up scales with the size of the list.
  - ✓ myList = ['A', 'B', 'C']
  - ✓ mySet = {'A', 'B', 'C'}
  - ✓ myDict = {'A': 5, 'B': 2, 'C': 7}
  - ✓ if 'A' in myList: # slow
  - ✓ if 'A' in mySet: # fast
  - ✓ if 'A' in myDict: # fast

10^{-9}	1 nanosecond	ns	One billionth of one second
10^{-6}	1 microsecond	µs	One millionth of one second
10^{-3}	1 millisecond	ms	One thousandth of one second

- timeit provides a simple way to time small bits of Python code.

https://docs.python.org/3.7/library/timeit.html

Online Resources for Python

- Free Online Course
  - Introduction To Python Programming. https://www.udemy.com/course/pythonforbeginnersintro
  - Introduction to Computer Science and Programming Using Python. https://www.edx.org/course/6-00-1x-introduction-to-computer-science-and-programming-using-python-3
- Free Online Books with Exercise
  - Google. https://developers.google.com/edu/python/
  - W3School. https://www.w3schools.com/python/default.asp
  - Python3 菜鸟教程. https://www.runoob.com/python3/python3-tutorial.html
  - Official document. https://docs.python.org/3/library/index.html
  - https://erlerobotics.gitbooks.io/erle-robotics-learning-python-gitbook-free/

Exercise on String

https://www.w3resource.com/python-exercises/string/

- 1. Write a Python program to calculate the length of a string.
- 4. Write a Python program to get a string from a given string where all occurrences of its first char have been changed to '$', except the first char itself.
- 5. Write a Python program to get a single string from two given strings, separated by a space and swap the first two characters of each string.

Exercise on Data Structure

https://www.runoob.com/python3/python3-dictionary.html

https://erlerobotics.gitbooks.io/erle-robotics-learning-python-gitbook-free/lists/exercises_list_and_dictionaries.html

Outline

- Introduction to Python
- Python Basics(syntax)
- Control flow and basic data structure
- Data Understanding
- Pandas and Numpy

Types of Data – Structure

- **Structured data** refers to any data that resides in a fixed field within a record.
  - ✓ Stored in tables with columns and rows: relational databases, spreadsheets, Pandas dataframe…
  - ✓ Most favorable for analysis
- **Semi-structured data** does not conform with table forms , but contains tags or other markers to separate semantic elements and enforce hierarchies of records and fields within the data.
  - ✓ Json, XML,HTML …
- **Unstructured data** does not have a pre-defined data model or is not organized in a pre-defined manner
  - ✓ 80% or even higher of data is Unstructured data : text, image, video, voice …

Types of Data – Label

- **Labeled data** is a group of samples that have been tagged with one or more labels, and labels are of our target output
  - ✓ Profile data of bank users with labels showing whether users default
  - ✓ Pictures of animals but come with labels showing animal species

- **Unlabeled data (most data)** are samples have not been tagged with one or more labels that is of our interests
  - ✓ Profile data of bank users
  - ✓ Pictures of animals

Types of Data

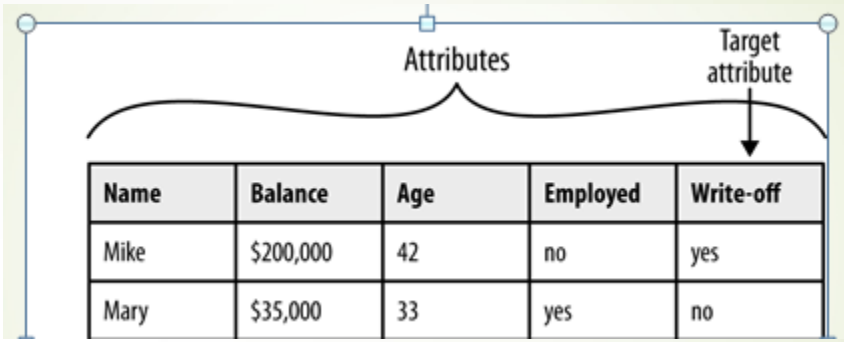
Positive

Negative

广汽集团携手腾讯发展智能汽车

深交所发函质疑大连友谊资产重组

Labeled



Name	Balance	Age	Employed	Write-off
Mike	$200,000	42	no	yes
Mary	$35,000	33	yes	no

Unstructured

Structured

广汽集团携手腾讯发展智能汽车

深交所发函质疑大连友谊资产重组

Name	Balance	Age	Employed
Mike	$200,000	42	no
Mary	$35,000	33	yes

Unlabeled

Types of Data

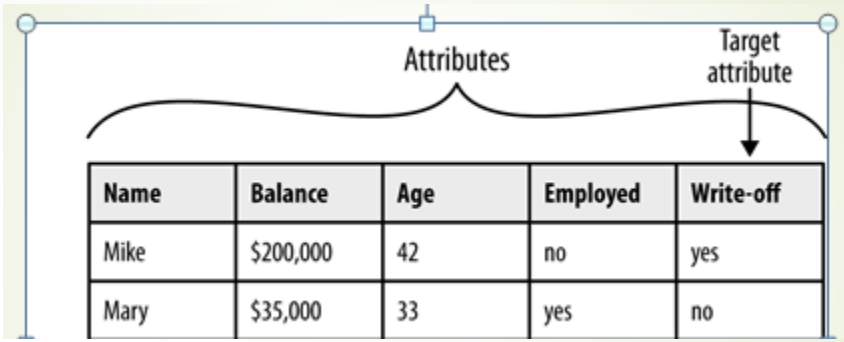
Positive

Negative

广汽集团携手腾讯发展智能汽车

深交所发函质疑大连友谊资产重组

Labeled



Name	Balance	Age	Employed	Write-off
Mike	$200,000	42	no	yes
Mary	$35,000	33	yes	no

Unstructured

Structured

广汽集团携手腾讯发展智能汽车

深交所发函质疑大连友谊资产重组

Name	Balance	Age	Employed
Mike	$200,000	42	no
Mary	$35,000	33	yes

Unlabeled

Data Labelling

- Collect labels from other sources
  - ✓ Proxy: Grade for student learning capacity, dishonest people for loan default
  - ✓ http://shixin.court.gov.cn/

- Human annotated
  - ✓ Crowd sourcing

- Rules
  - ✓ Be careful … …

Types of Data

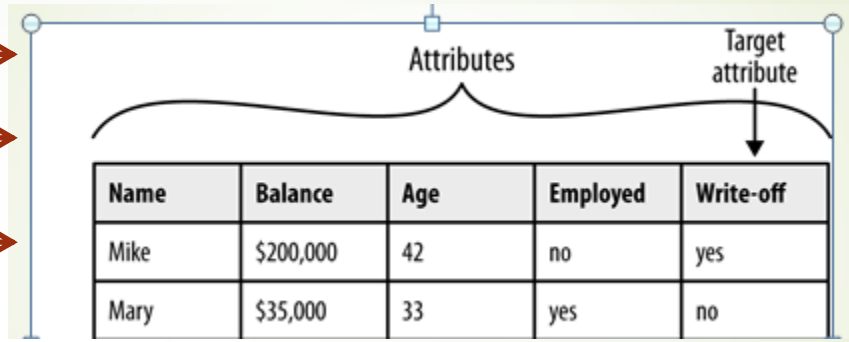
Labeled

Positive

Negative

广汽集团携手腾讯发展智能汽车

深交所发函质疑大连友谊资产重组



Name	Balance	Age	Employed	Write-off
Mike	$200,000	42	no	yes
Mary	$35,000	33	yes	no

Unstructured

Structured

广汽集团携手腾讯发展智能汽车

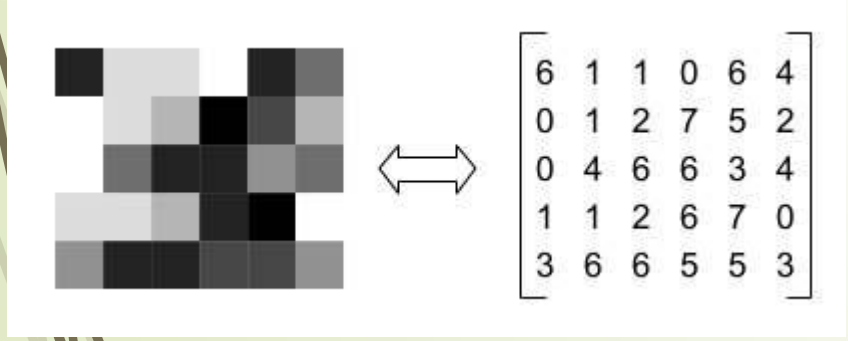
深交所发函质疑大连友谊资产重组

Name	Balance	Age	Employed
Mike	$200,000	42	no
Mary	$35,000	33	yes

Unlabeled

Converting to Structured Data

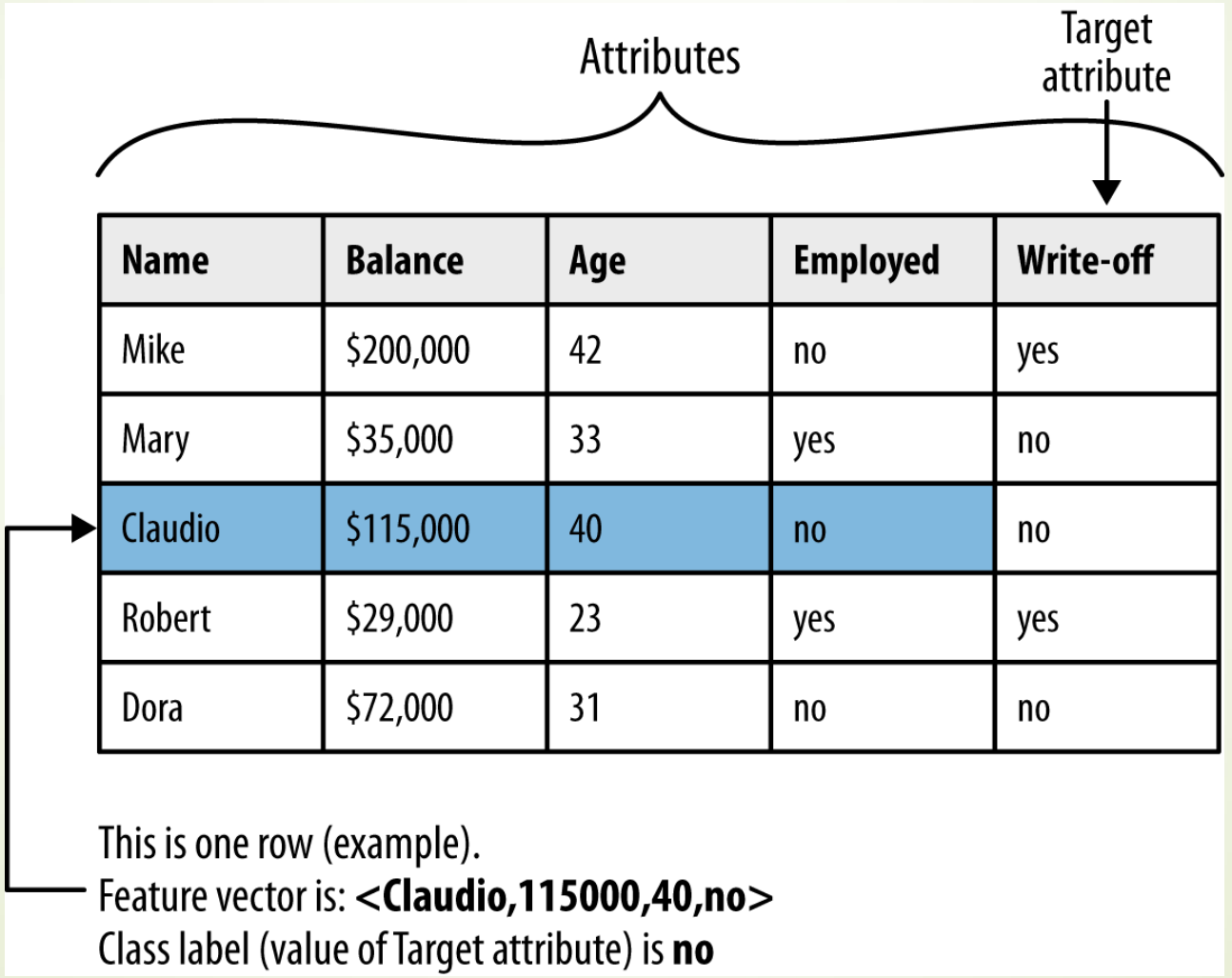
- Semi-structured data -> Structured data
  - ✓ Package json can parse json into data stored in Python data structure
  - ✓ Package beautifulsoup/lxml can parse XML,HTML in structured way
- Unstructured data -> Structured data
  - ✓ Text data: vector space model
  - ✓ Image data: matrix element is pixel intensity



doc1: I like football
doc2: John likes football
doc3: John likes basketball

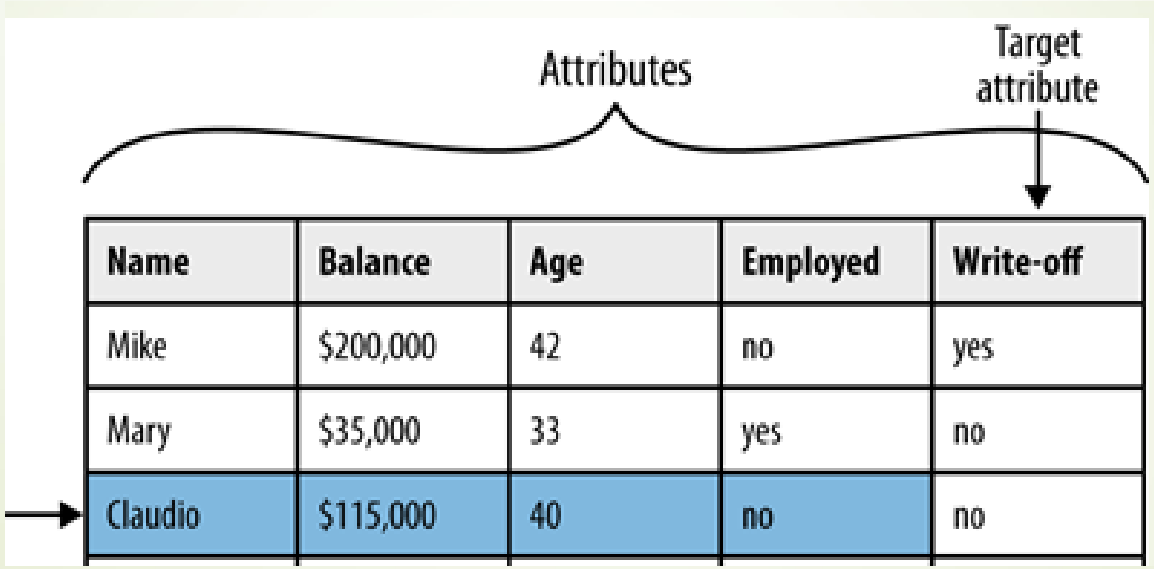
	doc1	doc2	doc3
I	1	0	0
like	1	0	0
John	0	1	1
likes	0	1	1
football	1	1	0
basketball	0	0	1

Structured Data



Different Names of The Same Thing

- Table = dataset = worksheet
- Rows = examples = cases = instances = records
- Columns = features = attributes = fileds = explanatory variable = predictors
- Target attribute = target variable = dependent variable = label



Name	Balance	Age	Employed	Write-off
Mike	$200,000	42	no	yes
Mary	$35,000	33	yes	no
Claudio	$115,000	40	no	no

Types of Attributes

- **Categorical (nominal)** : finite names
  - ✓ Color, gender, country
- **Ordinal:** with finite values which can have orders
  - ✓ Grade of students : A+, A, B ,C …
- **Numerical :** values are numbers
  - ✓ Continuous: floating number; Discrete: a subset of integer values
- **Interval:** value with fixed size of interval and difference between values is meaningful.
  - ✓ **Date, Date time** , temperature

Other Data – Transaction Data

- Each record involves in a set of items

TID	Items
1	Bread, Coke, Milk
2	Beer, Bread
3	Beer, Coke, Diaper, Milk
4	Beer, Bread, Diaper, Milk
5	Coke, Diaper, Milk

Other Data – Graph data

- Social graph can be represented and stored as triplets (user1,user2, type)
  - ✓ Graph theory: shortest path, centrality, community

Pandas

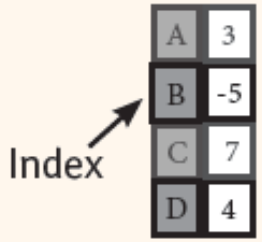
- **Pandas** provide data structures designed to make working with structured (tabular, multidimensional, potentially heterogeneous) and time series data both easy and intuitive.
  - ✓ The two primary data structures of pandas, **Series** (1-dimensional) and **DataFrame** (2-dimensional), handle the vast majority of typical use cases
  - ✓ The name Pandas is derived from the word **Pan**el **Da**ta – an Econometrics from Multidimensional data.
  - ✓ We focus on **DataFrame** which can handle **tabular data** with heterogeneously-typed columns, as in a Excel spreadsheet.

Person ID	Age	Gender	Income	Balance	Mortgage payment
123213	32	F	25000	32000	Y
17824	49	M	12000	-3000	N
….	….	….	….	….	….

Series and Dataframe

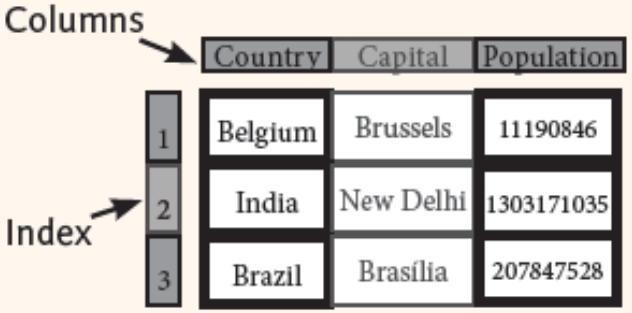
Series

A **one-dimensional** labeled array capable of holding any data type




```
>>> s = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd'])
```

DataFrame



A **two-dimensional** labeled data structure with columns of potentially different types

```
>>> data = {'Country': ['Belgium', 'India', 'Brazil'],
            'Capital': ['Brussels', 'New Delhi', 'Brasília'],
            'Population': [11190846, 1303171035, 207847528]}

>>> df = pd.DataFrame(data,
                      columns=['Country', 'Capital', 'Population'])
```

Source:http://www.kdnuggets.com/2017/01/pandas-cheat-sheet.html

Gain an overview

- **df.head():** Get an overview of top 5 lines
- **df.info():** Get an (techinque) summary of a DataFrame.
- **df.describe():** Get descriptive statistics that summarize the central tendency, dispersion and shape of a dataset's distribution, excluding NaN values.
- **df.shape:** Return tuple representing the dimensionality of the DataFrame.
- **df.index**: Return the index

```
In [3]: df.head()
```

Out[3]:

Date	Open	High	Low	Close	Adj Close	Volume
2017-01-03	20.549999	20.879999	20.549999	20.730000	20.047922	21701669
2017-01-04	20.740000	20.950001	20.450001	20.850000	20.163973	33155480
2017-01-05	20.850000	21.230000	20.780001	20.930000	20.241341	31012563
2017-01-06	20.940001	21.040001	20.610001	20.639999	19.960882	23591954
2017-01-09	20.600000	20.750000	20.530001	20.660000	19.980225	15095445

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 164 entries, 2017-01-03 to 2017-09-01
Data columns (total 6 columns):
Open          164 non-null float64
High          164 non-null float64
Low           164 non-null float64
Close         164 non-null float64
Adj Close     164 non-null float64
Volume        164 non-null int64
dtypes: float64(5), int64(1)
memory usage: 14.0 KB
```

Index in Pandas

- Both dataframe and series can be indexed by non-numeric values (datetime)
  - ✓ use **loc[]** to access data based on label/index
  - ✓ Both will create a default integer index as integer position, use **loc[]** to access
  - ✓ Both can be accessed by integer position as in list, use **iloc[]**to access
  - ✓ Series index each element in a 1-d list, Dataframe index each row
  - ✓ Can be viewed by s.index or df.index

column

Index
0 (a)
1 (b)
2 (c)

Person ID	Age	Gender	Income	Balance	Mortgage payment
123213	32	F	25000	32000	Y
17824	49	M	12000	-3000	N
….	….	….	….	….	….

row

Create and Index Pandas Series

- Pandas series can be created from list, tuple, numpy array, dict ….
- By default, s[index_val] returns the value in Serie s with index value "index_val"
- Each row/column of dataframe is a series
- s2.loc[[101,102,103]] equals to s2.iloc[:3] (s2.loc[[101:103] also work but tricky)

```
In [39]: s1=pandas.Series(['a','b','c','d'])

In [40]: s1.index
Out[40]: RangeIndex(start=0, stop=4, step=1)

In [41]: print(s1)
0    a
1    b
2    c
3    d
dtype: object
```

```
In [52]: s2=pandas.Series(['a','b','c','d'],index=[101,102,103,104])

In [53]: s2.index
Out[53]: Int64Index([101, 102, 103, 104], dtype='int64')
In [54]: print (s2)
101    a
102    b
103    c
104    d
dtype: object
```

```
In [59]: print(s1.iloc[0],s1[0],s1.loc[0],s2.iloc[0],s2[101],s2.loc[101])
a a a a a a
```

Create and Index Pandas Dataframe

- Pandas dataframe can be created from numpy array , dict, or outside file..
- df.loc[row_indexer,column_indexer] based on index/label
- df.iloc[row_indexer,column_indexer] based on position
- df.loc[['a','b'],['Age','Gender']] = df.iloc[:2,1:3]
- By default, df[label] returns the series of selected **column**, such as df['income'] = df.loc[:,['income']]

df.loc[:,['Age','Gender']],or df.iloc[:,1:3] or df['Age','Gender']

df.loc[:,['Age','Gender']] or df.iloc[:,:2] or df[:2]

Index	Person ID	Age	Gender	Income	Balance	Mortgage payment
a	123213	32	F	25000	32000	Y
b	17824	49	M	12000	-3000	N
c	….	….	….	….	….	….

row

Selecting on Conditions

- Dataframe/Series can also be selected by logic conditions (like SQL)

```
high2low=df[df['Open']<df['Close']][['Open','Close']]
high2low.head()
```

Date	Open	Close
2017-01-03	20.549999	20.730000
2017-01-04	20.740000	20.850000
2017-01-05	20.850000	20.930000
2017-01-09	20.600000	20.660000
2017-01-13	21.000000	21.809999

```
high2low.shape
```

```
(78, 2)
```

```
len(high2low)
```

```
78
```

Apply functions

- Dataframe/Series have built-in functions to process the whole data structure
  - ✓ Descriptive statistics, mean(), max(), std()
  - ✓ Common operator like +,-,,/ can be applied to all data element directly
  - ✓ Apply(func) can applying common functions to the to all data element

```
In  [11]: df['Open'].mean()
Out[11]: 21.52158540853658

In  [12]: df['Open'].std()
Out[12]: 1.7205970417676413

In  [13]: df['Open']+=5

In  [14]: df['Open'].mean()
Out[14]: 26.521585408536595
```

```
In  [17]: def minus5(x):
              return x-5
          df['Open']=df['Open'].apply(minus5)
          df['Open'].mean()
Out[17]: 21.52158540853658
```

Group by functions

- Group also borrows from SQL
  - ✓ Especially useful for grouping by category variables

```
In [20]: import numpy as np
df = pd.DataFrame({'Gender' : ['M', 'F', 'M', 'F',
                               'M', 'M', 'F', 'F'],
                   'XXX' : ['one', 'one', 'two', 'three',
                            'two', 'two', 'one', 'three'],
                   'Income' : np.random.randn(8)+1000,
                   'Balance' : np.random.randn(8)+10000})
df.head()
```

Out[20]:

	Balance	Gender	Income	XXX
0	9999.042622	M	999.713548	one
1	10001.985779	F	1000.319083	one
2	10002.154422	M	998.904281	two
3	9998.237199	F	1000.385490	three
4	10001.720845	M	998.628000	two

```
In [25]: df.groupby('Gender').mean()
```

Out[25]:

Gender	Balance	Income
F	9999.762972	999.839746
M	10000.793025	999.030950

```
In [28]: df.groupby('Gender')['Income'].mean()
```

```
Out[28]: Gender
F    999.839746
M    999.030950
Name: Income, dtype: float64
```

Add Columns and Rows

- Dataframe can easily add rows and columns
  - ✓ df['new_col'] = XXX df.loc['new_index'] = XXX

```
In [8]:
   ...: df['12h'] = df['Open']<df['Close']

In [9]: df.head()
Out[9]:
         Date       Open       High        Low      Close  Adj Close  \
0  2017-01-03  20.549999  20.879999  20.549999  20.730000  20.047922
1  2017-01-04  20.740000  20.950001  20.450001  20.850000  20.163973
2  2017-01-05  20.850000  21.230000  20.780001  20.930000  20.241341
3  2017-01-06  20.940001  21.040001  20.610001  20.639999  19.960882
4  2017-01-09  20.600000  20.750000  20.530001  20.660000  19.980225

     Volume    12h
0  21701669   True
1  33155480   True
2  31012563   True
3  23591954  False
```

Numpy array

- Numpy has array similar to dataframe but can only handle homogeneous array of fixed-size items
  - ✓ After you handle non-numeric values , you can generate corresponding numpy ndarray
  - ✓ Useful for adopting most machine learning tasks
  - ✓ Various linear algebra computation can be done on numpy ndarray

```
In  [50]: df.values

Out[50]: array([[  2.05499990e+01,   2.08799990e+01,   2.05499990e+01,
                   2.07300000e+01,   2.00479220e+01,   2.17016690e+07],
                [  2.07400000e+01,   2.09500010e+01,   2.04500010e+01,
                   2.08500000e+01,   2.01639730e+01,   3.31554800e+07],
                [  2.08500000e+01,   2.12300000e+01,   2.07800010e+01,
```

```
In  [49]: type(df.values)

Out[49]: numpy.ndarray
```

Tips and More

- Index is usually numbered starting with 0, max index will be len(data) -1
- End of slicing index is usually not included, s[:3] include s[0],s[1],s[2]
- Dataframe is powerful, search the buil-in functions first
  (merge, concat, handle missing value , time series)
- Try to apply functions with apply rather than using for-loops
- For more tutorials on Pandas
  - ✓ Official: http://pandas.pydata.org/pandas-docs/stable/index.html
  - ✓ Hands-on code: https://github.com/jvns/pandas-cookbook
- For more tutorials on Numpy
  - ✓ Official: https://docs.scipy.org/doc/numpy-dev/user/quickstart.html
  - ✓ Hands-on code: https://github.com/rougier/numpy-tutorial