ΠΑΝΕΠΙΣΤΗΜΙΟ
ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

Introduction to Information Systems and Applications

Course Unit 2: Data processing with python

M. Tzagarakis, V. Daskalou

School of Business Administration

Department of Economics

Unit Scope

- Make an introduction to the features of the python language
- Introduce the basic capabilities of programming languages for data processing

Unit contents

1. Characteristics of python
2. Programming with python
    1. Basics:

       Calculations and variables, Strings, Flow control, Composite types (lists, tuples, dictionaries), Functions, Files
    2. Manipulate CSV files
    3. Figures and plots
    4. Statistics

The python language

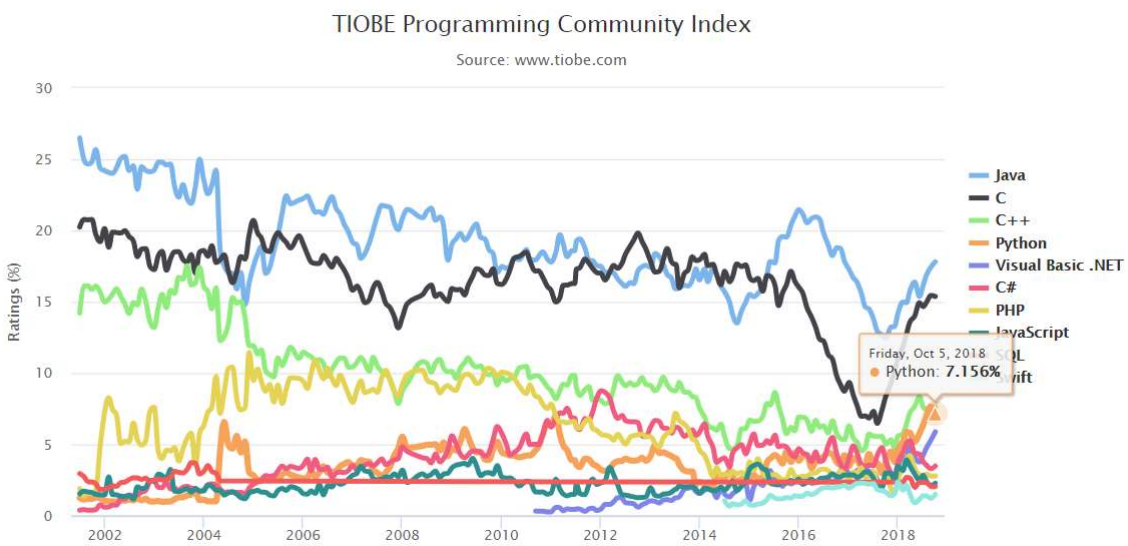
History of Python

- Created by Guido van Rossum in 1989 (name inspired by Monty Python)
- 2 versions:
  - Python 2.0 (October 2000) last versions 2.6 & 2.7
  - Python 3.0 (December 2008): not completely backward-compatible

Guido van Rossum

Πηγή : https://en.wikipedia.org/wiki/Guido_van_Rossum

python: Importance



Top-10 programming languages (Oct-2018)

By TIOBE Software B.V. [CC BY-SA 4.0 (http://creativecommons.org/licenses/by-sa/4.0)], via Wikimedia Commons
Source: https://www.tiobe.com/tiobe-index//

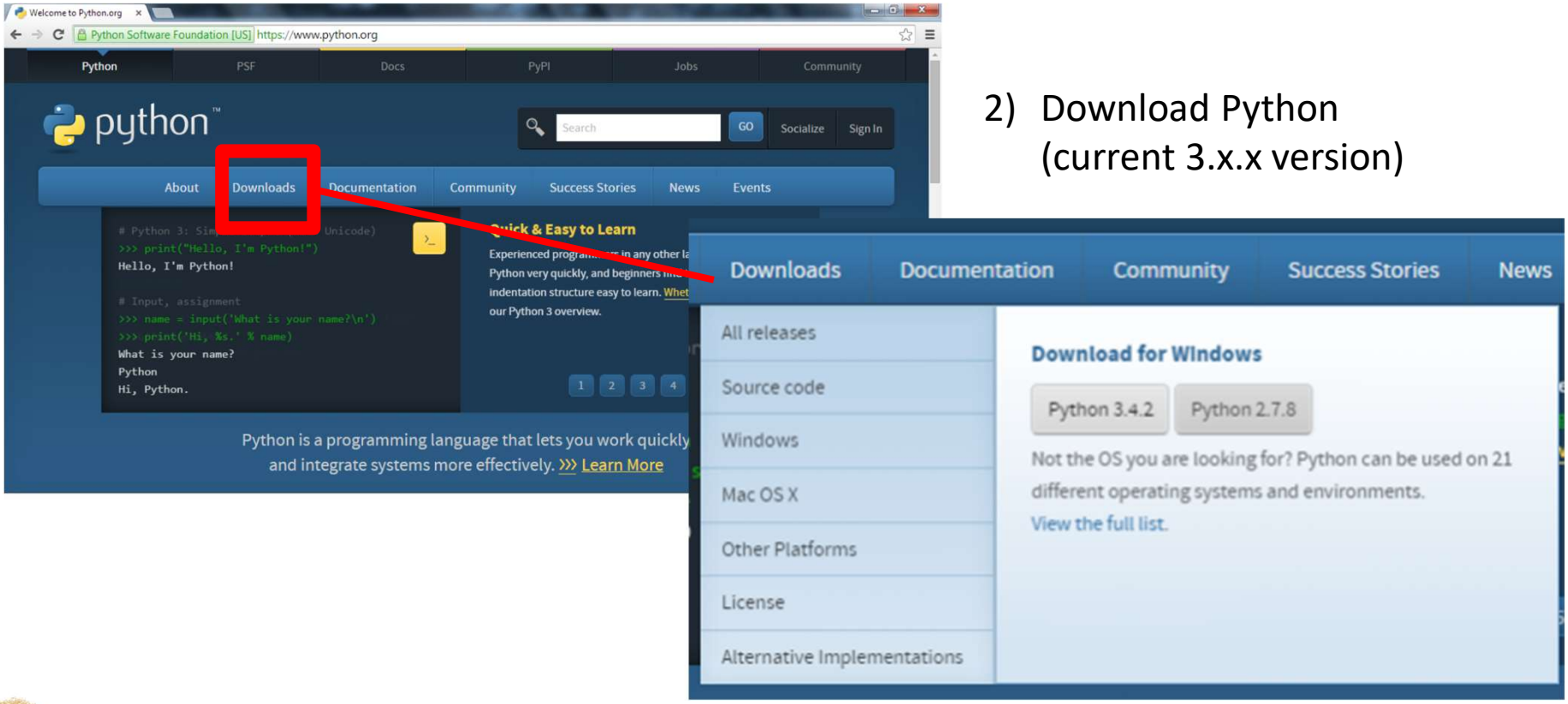
- #4 in top-10 programming languages
- Important for web, dbs & academic computing (source)
- the Most Popular Introductory Teaching Language at Top U.S. Universities (source)

Main characteristics

- Open source
- High level programming: Use natural language terms (English) ->easy to understand
- Compatible in different environments (OS)
- Philosophy: *"there should be one—and preferably only one—obvious way to do”*
  instead of "there is more than one way to do it" (Perl language)

Installation: base python (1)

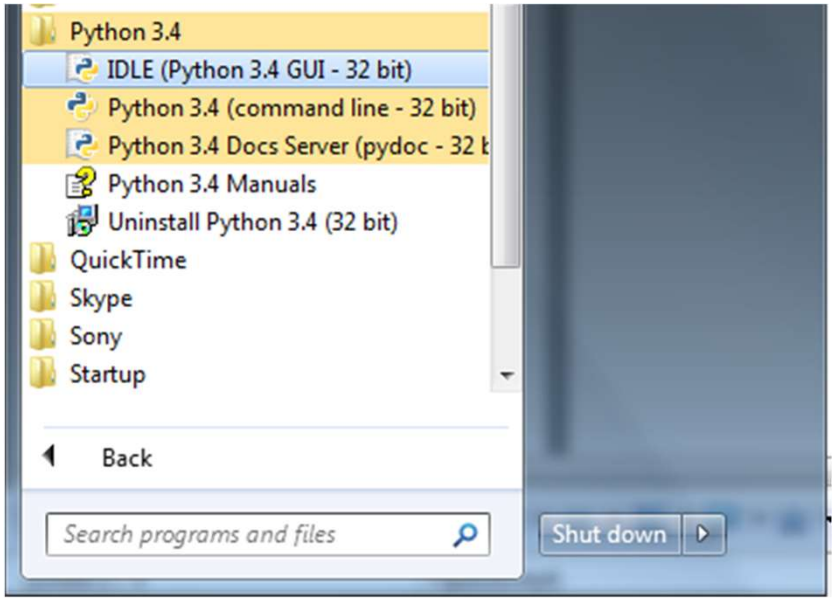
1) Visit http://www.python.org/

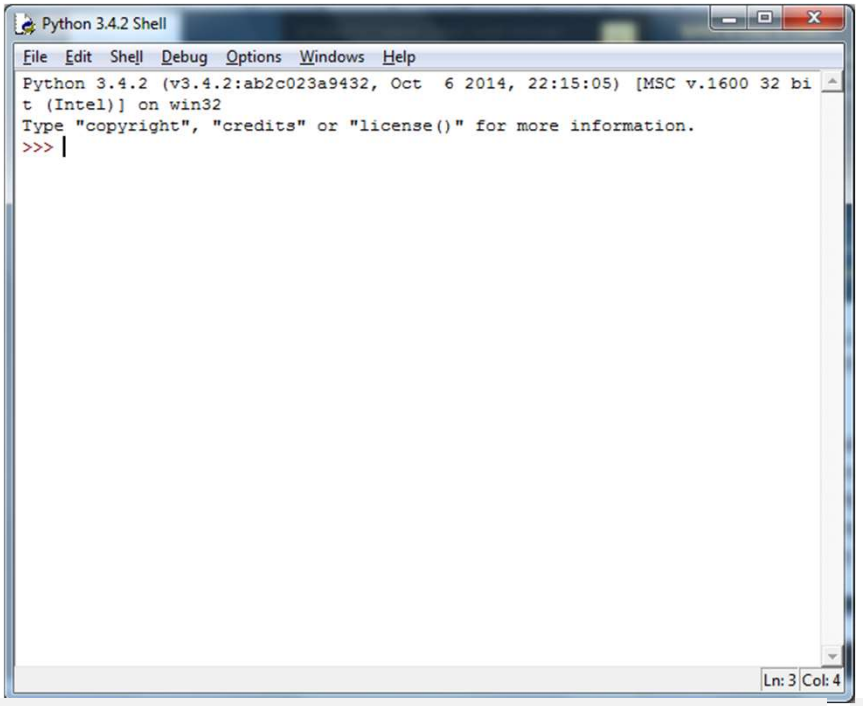


2) Download Python (current 3.x.x version)

Installation: base python (2)

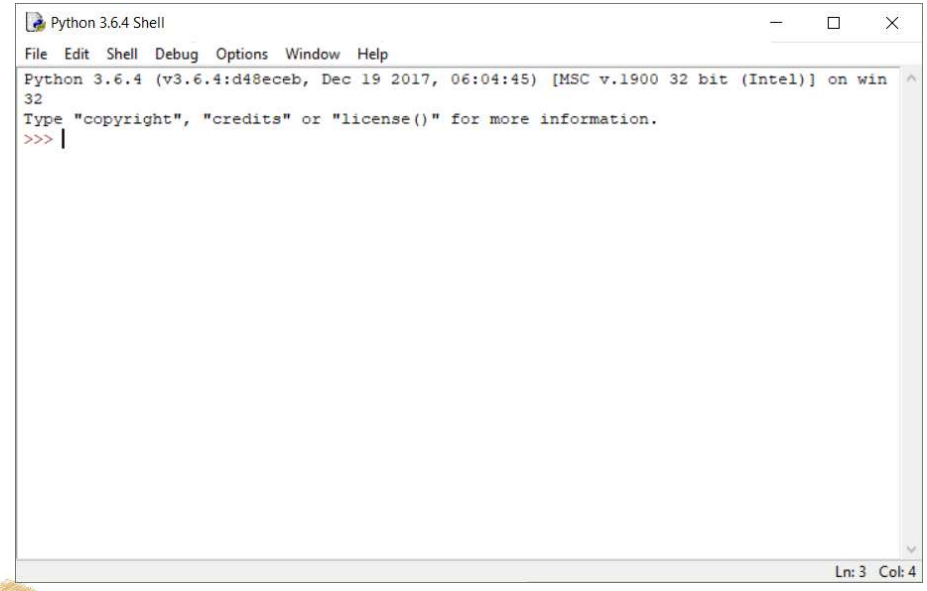
3) Download and install following the wizard. **IMPORTANT**: Click "Add python to path"
4) 2 ways to run a Python program:
   - Source file .py
   - Interactive interpreter prompt:
     - From Start Menu, open IDLE as follows: Start → All Programs → Python 3.x → IDLE (Python GUI)

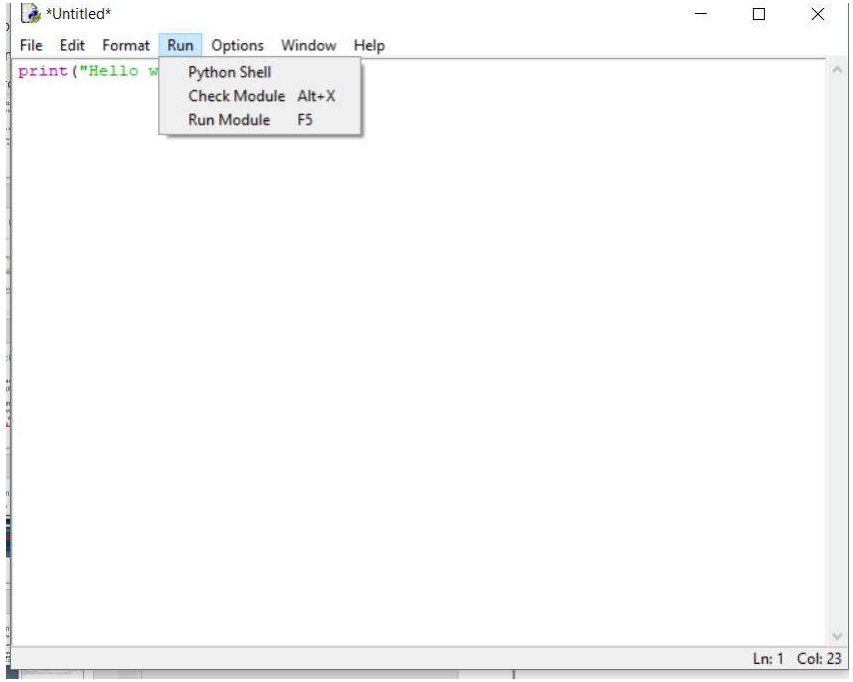




Python IDLE

- "Integrated DeveLopment Environment" for Python.
- A software package that lets us write Python commands and edit and run Python programs.
- Python Shell:
- Helps writing python programs easily and create source files:
  - File->New->Save->Run->Run Module(F5)





Anaconda python

- Python distribution from Continuum Analytics
- Advantages:
  - Pre-installed popular add-in packages
  - Integrated development environment: Spyder
- Download:
  - https://www.anaconda.com/download/
  - Anaconda 5.3 For Windows Installer->Python 3.7 version->Download
  - Double click the downloaded .exe and follow the instructions

Calculation and variables

Calculation and operators

```
>>> (2+5)*5
35
>>> 2**2
4
>>> 14/4
3.5
>>> 14//4
3
>>> 14%4
2
>>> 2**1000
10715086071862673209484250490600018105614048117055336074437503883703510511249361224931983788156958581275946729175531468251871452856923140435984577574698574803934567774824230985421074605062371141877954182153046474983581941267398767559165543946077062914571196477686542167660429831652624386837205668069376
```

Operator	Description	Example
+ Addition	Adds values on either side of the operator.	a + b = 31
- Subtraction	Subtracts right hand operand from left hand operand.	a – b = -11
* Multiplication	Multiplies values on either side of the operator	a * b = 210
/ Division	Divides left hand operand by right hand operand	b / a = 2.1
% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a = 1
** Exponent	Performs exponential (power) calculation on operators	a**b =10 to the power 20
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity):	9//2 = 4 and 9.0//2.0 = 4.0, -11//3 = -4, -11.0//3 = -4.0

Variables and assignment

Variable:

- reserved memory locations to store values
- when you create a variable you reserve some space in memory
- a name to reference that space

Assignment:

- `vrbl=5`
  use = to store values in a variable, not equal
- `a=a+a`
  the use of a variable name on the right of the assignment operator = refers to the value stored in the variable

Variables

- Names:
  - letters, digits, or underscores _, always begin with a letter
- Reserved words
- Case sensitive
- Naming conventions:
  https://en.wikipedia.org/wiki/Naming_convention_(programming)#Python_and_Ruby
  lowercase_separated_by_underscores

Basic variable types

String (str): (in single or double quotes)

```
mystr='Hello Nikos',
grGM="Goodmorning",
yourPhone='2610459220'
```

Integer (int):

```
a=6, b=1234, c=-567
```

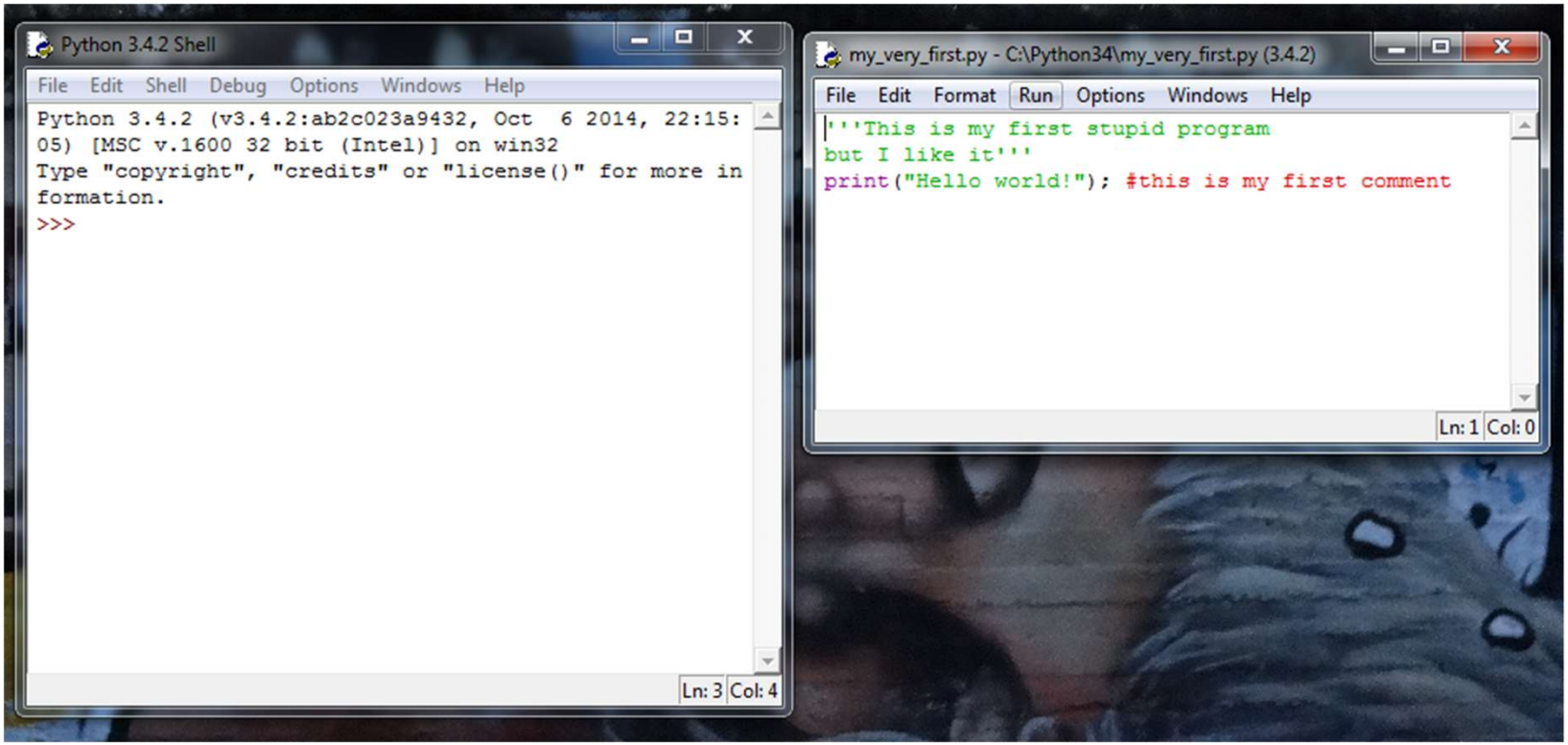
Floating point (float):

```
x=6.2, y=52.3E-4, z=-567.56789
```

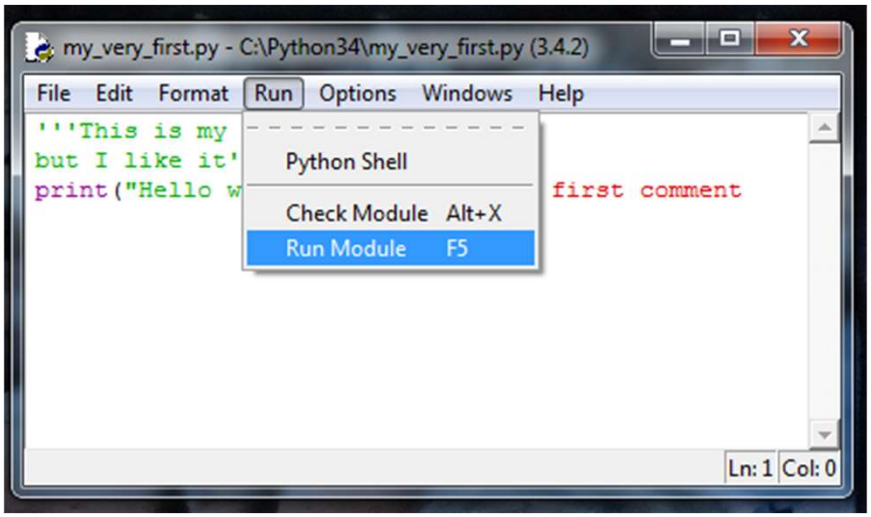
Boolean: `True, False`

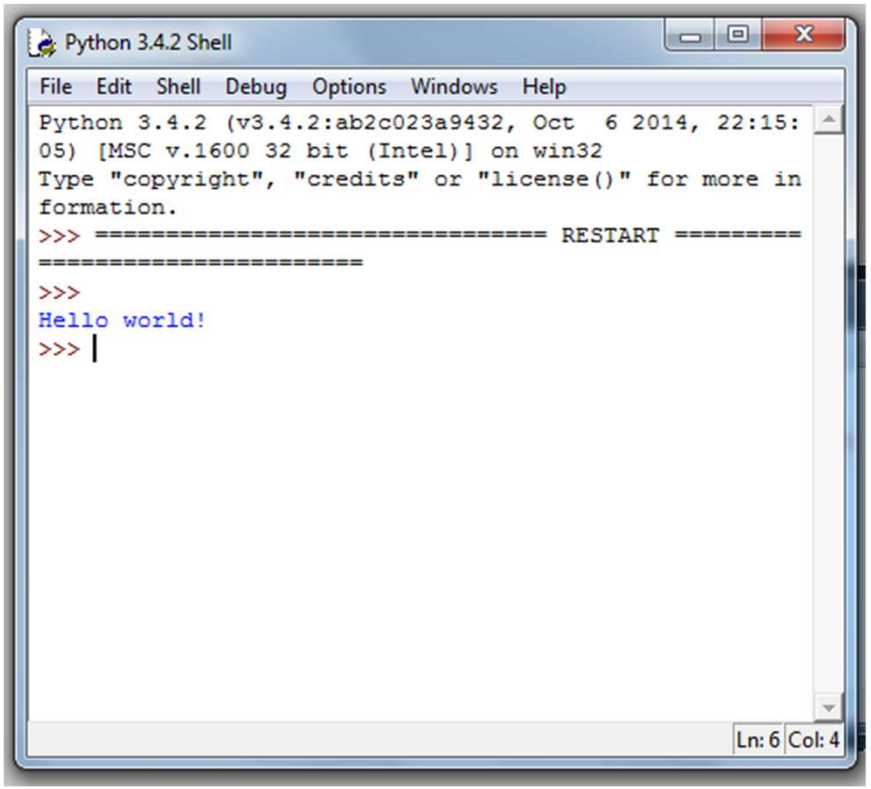
My first program in IDLE

IDLE->File->New File



My first program in IDLE: Execute

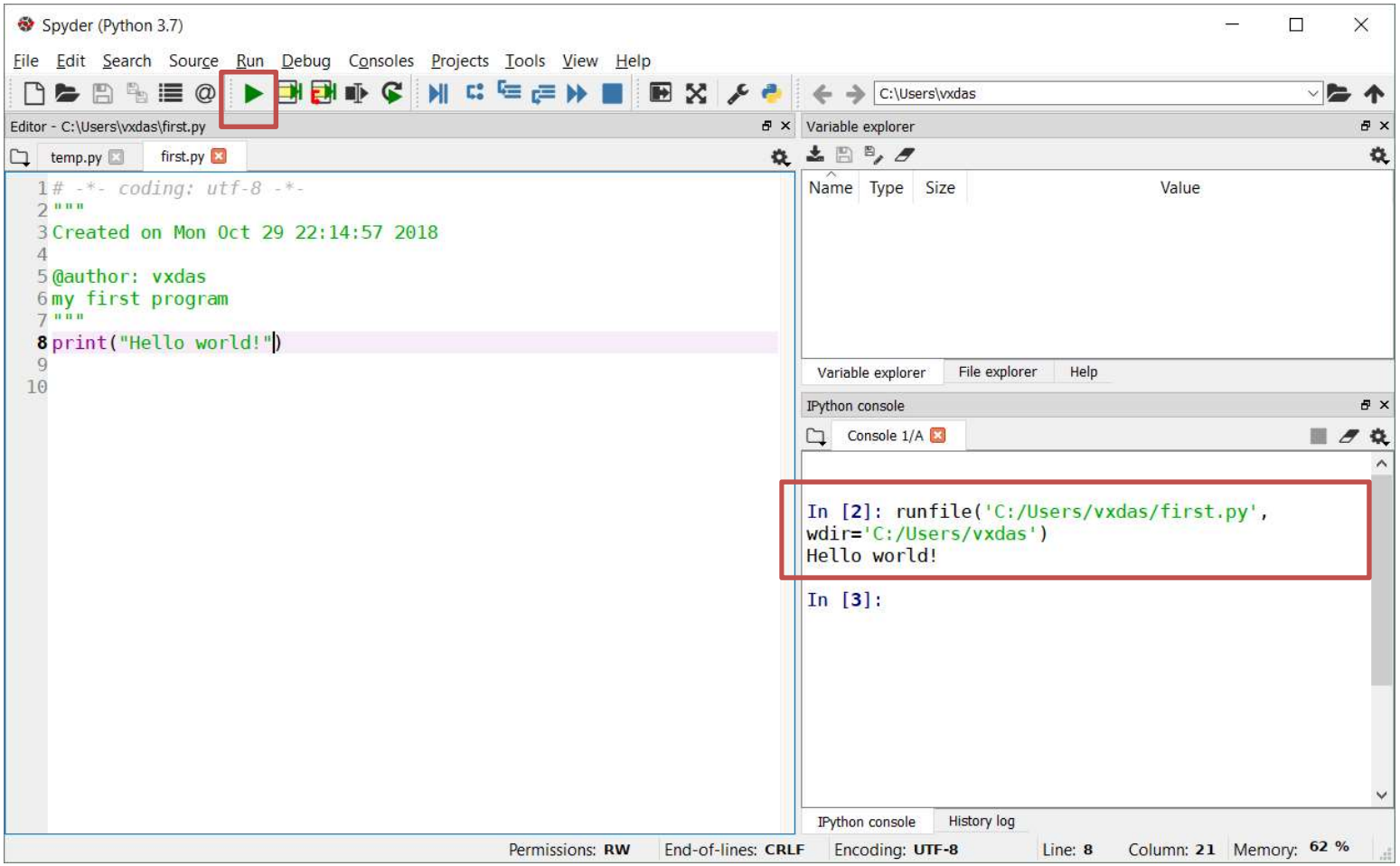




My first program in IDLE

1. At IDLE Shell choose *File->New file*
2. The window of IDLE editor opens
3. We write our first program with python commands
4. We save our program with *File->Save*
5. We execute the program from *Run->Run Module*
6. The IDLE Shell window prints *RESTART* and starts the execution of our programs

My first program: Anaconda Spyder



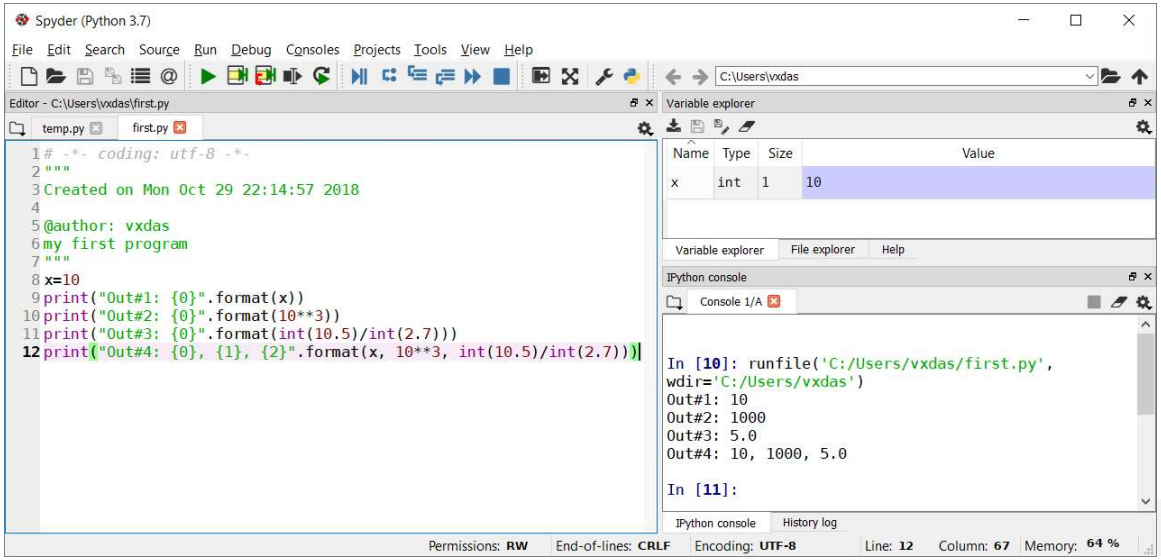
Numbers

Numbers (I)

Integers

Understand " ".format() function
https://www.geeksforgeeks.org/python-format-function

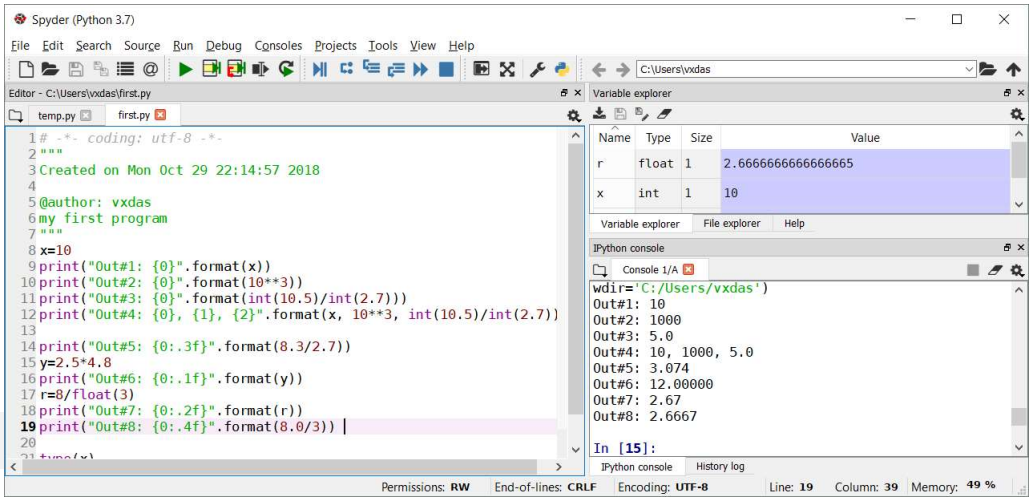
```
x=10

print("Out#1: {0}".format(x))

print("Out#2: {0}".format(10**3))

print("Out#3: {0}".format(int(10.5)/int(2.7)))

print("Out#4: {0}, {1}, {2}".format(x, 10**3, int(10.5)/int(2.7)))
```



Numbers (II)

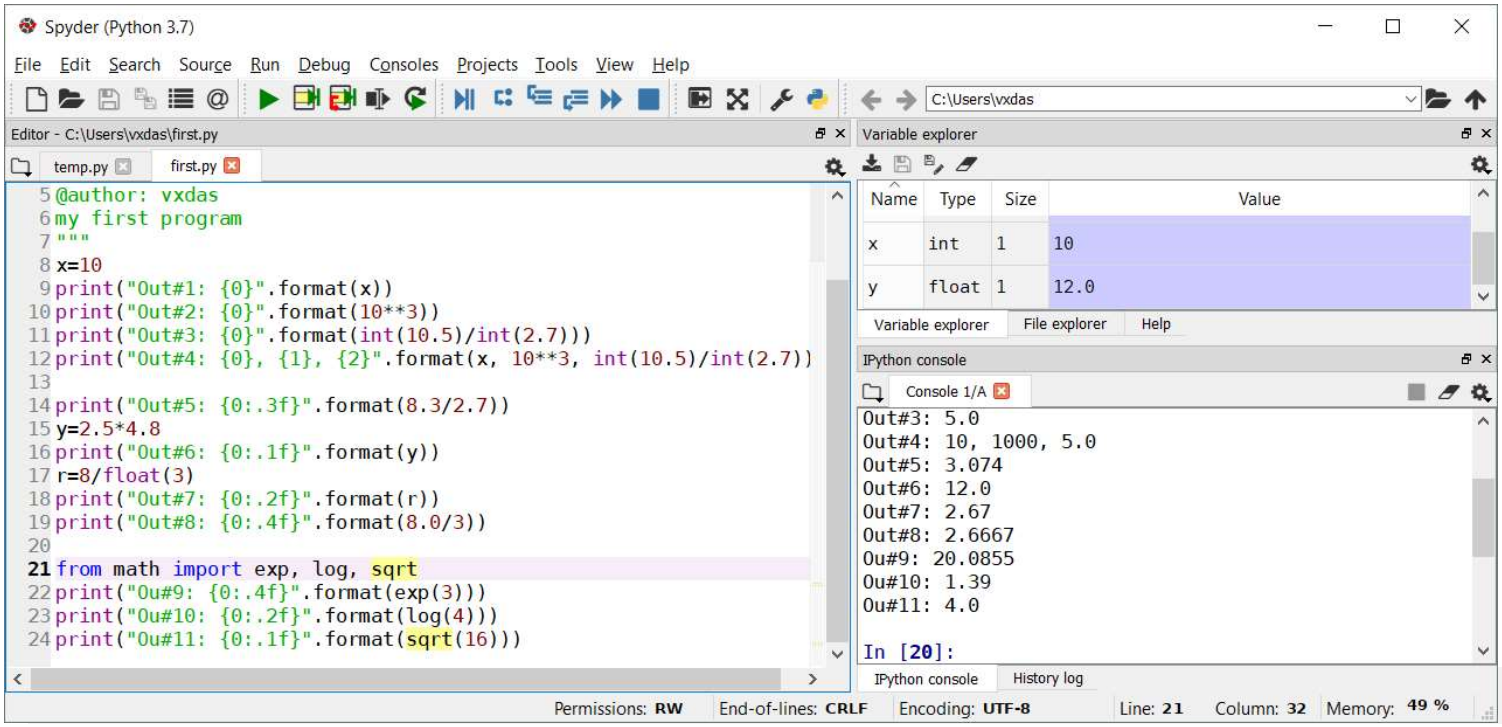
- Floating-point numbers

```
print("Out#5: {0:.3f}".format(8.3/2.7))

y=2.5*4.8

print("Out#6: {0:.1f}".format(y))

r=8/float(3)

print("Out#7: {0:.2f}".format(r))

print("Out#8: {0:.4f}".format(8.0/3))
```



`math` module

```
from math import exp, log, sqrt

print("Ou#9: {0:.4f}".format(exp(3)))

print("Ou#10: {0:.2f}".format(log(4)))

print("Ou#11: {0:.1f}".format(sqrt(16)))
```



User input with input() function

```
>>> a=input("Give me first #: ")
Give me first #: 5
>>> b=input("Give me second #: ")
Give me second #: 10
>>> a+b
'510'
>>> int(a)+int(b)
15
>>> x=float(input(Give me a real: '))
Give me a real: 567.1234
>>> print(x)
567.1234
```

input(prompt)

- Parameter: prompt string
- Return value: string
- Integer input:
  `int(input("Give an integer: "))`
- Real input:
  `float(input("Give a real: "))`

Exercises

- Write a program that asks the user for 5 real numbers and calculates their average
- Write a program requesting the user the coefficients a, b, c of the trinomial ($a \cdot x^2 + b \cdot x + c$) and calculate its value for a specific x that will be given by the user
- Write a program that asks the user for the values in meters of the base and height of a triangle and calculates its surface
- A circular fountain is located in a rectangular courtyard. Write a program to ask the user for the width and length of the yard, the diameter of the fountain (in meters) and calculate how much should we pay in order to fully cover the courtyard with tiles which have a specific dimension in cm (25x35cm) and cost 15 euros each one
- An American traveler comes to Greece with a specific currency in dollars. Write a program that asks the user for the amount and counts to how many euros can be converted (find the current parity in the Internet and use it as a constant, ignoring the cents) and how many banknotes of 50, 20, 10 and 5 euros the traveler will get from the exchange.

Strings

Strings

```
>>> 'very ' + 'hot'
'very hot'
>>> 3*'very ' + 'hot'
'very very very hot'
>>> '7'+'2'
'72'
>>> type('dog')
<class 'str'>
>>> type('7')
<class 'str'>
>>> type(7)
<class 'int'>
```

```
>>> justaTest = '''Say,
"I'm in!"
This is line 3'''
>>> print(justaTest)
Say,
"I'm in!"
This is line 3
>>>
```

- String: Alphanumeric in single or double quotes
- Use triple single or triple double quotes for writing in multi-line strings
- Use + for string concatenation

String operations(1)

- `yourString.upper()` – uppercase letters
- `yourString.lower()` – lowercase letters
- `yourString.capitalize()` – capitalize only the first string letter
- `yourString.title()` – capitalize the first letter of every word
- `yourString.replace(x,y)` - replace **x** with **y**
- `len(yourString)` – length of string

More string functions:

- https://docs.python.org/3.4/library/stdtypes.html#string-methods

String operations: example

```
>>> yourString='The answer to the ultimate question of life, the universe and
    everything is 42.'
>>> yourString.upper()
'THE ANSWER TO THE ULTIMATE QUESTION OF LIFE, THE UNIVERSE AND EVERYTHING IS 42.'
>>> yourString.lower()
'the answer to the ultimate question of life, the universe and everything is 42.'
>>> yourString.capitalize()
'The answer to the ultimate question of life, the universe and everything is 42.'
>>> yourString.title()
'The Answer To The Ultimate Question Of Life, The Universe And Everything Is 42.'
>>> yourString.replace('a','A')
'the Answer to the ultimAte question of Life, the Universe And Everything is 42.'
```

Strings: more operations

myStr	T	h	e		a	n	s	w	e	r		i	s		4	2
index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Every string is an *array of characters* (index starts at 0)

- `myStr[x]` - The xth character from the start (e.g `myStr[0]` is the first char)
- `myStr[start:stop]` - substring from start till stop-1
- `myStr[start:]` – substring from start till the end
- `myStr[:stop]` – substring from start till stop-1
- `myStr[:]` – the whole string
- `myStr[-x]` – The xth character from the end
- `myStr[-x:]` – the last x characters
- `myStr[:-x]` – the whole string without the last x characters

ΠΑΝΕΠΙΣΤΗΜΙΟ
ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

Introduction to Information Systems and Applications

Course Unit 2: Data processing with python

M. Tzagarakis, V. Daskalou

School of Business Administration

Department of Economics

Flow control

Flow control with `if`

```
if <condition>:
        # block of commands
else:
        # block of commands
```

Comparison operators:	`==`
	`!=`
	`>, >=`
	`<, <=`

Logical operators:	`not`, π.χ. not(a) `and`, π.χ. (a and b) `or`, π.χ. (a or b)
Member operators	`in, not in`

```
grade= int(input("Give your grade: "))
if grade>= 5:
    print("Pass :-)")
else:
    print("Fail :-(")
```

Attention: The indent in Python is important! It specifies the block of commands

if-elif-else

```
if <condition>:
    <block of commands>
elif <condition>:
    <block of commands>
else:
    <block of commands>
```

```
weather = input("How is the weather today? ")
if weather == "rainy":
    print("Take your umbrella")
elif weather == "cold":
    print("Take your jacket")
elif weather == "sunny":
    print("Wear your sun cream")
else:
    print("Have a nice day!")
```

for loops

```
for <variable> in <range>:
        <block of commands>

for counter in range(5):
        print("hello world")
print("outside for")
```

Tip: We use *for* loop when the number of iterations is known

`range(from, to, step)`

- `from, step`: optional
- `to`: obligatory
- `from, to, step`: **integers**

Examples of range():

- `range(10)`: [0,1,2,3,4,5,6,7,8,9]
- `range(1, 7)`: [1,2,3,4,5,6]
- `range(0, 30, 5)`: [0,5,10,15,20,25]
- `range(5, -1, -1)`: [5,4,3,2,1,0]

Example use of `for`

```
# calculate average of integers from start to stop-1
start=5
stop=11
#we dont use variable name sum, since it is a
reserved word
sumx=0
n=0
for x in range(start,stop):
  n=n+1
  print("x:{0} sum:{1} n{2}:".format(x,sumx,n))
  sumx=sumx+x
print("avg:", sumx/n) #outside for
```

Example use of `for` and `if`

```
# calculate average of even integers from start to stop-1
start=5
stop=100
sumx=0
n=0 #number of even numbers
for x in range(start,stop):
    if x%2==0 : # a number is even when its mod with 2 is 0
        sumx=sumx+x
        n=n+1
        print("x:{0} sum:{1} n{2}:".format(x,sumx,n))
print("avg :",sumx/n)
```

`while` loop

```
while <condition>:
        <command1>
        <command2>
[else:
        <command1 when loop ends>]
```

Tip: We use *`while`* when we don't know the number of iterations and we want to control the condition

```
#stops only if the user gives no
msg=input("Give a string: ")
while msg != 'no':
    print(msg*3+'!!!')
    msg=input("Give a string: ")
else:
    print("Thank you!")
```

Example use of `while` and `if`

```
# a guess game: user tries to guess the number
number = 23
running = True
while running:
    guess = int(input('Give a number: '))
        if guess == number:
                print('Congrads! You guess it!.')
                running = False # while stops here
        elif guess < number:
                print('No, is greater.')
        else:
                print('No, is smaller.')
else:
    print('Loop terminated.')
```

while loop with break, continue

```
# this runs till you give quit and check input length
while True:
  s = input('Give something : ')
  if s == 'quit':
      break # terminates the loop
  if len(s) < 3:
      print('to small!')
      continue #skips the rest of command and go to while
  print('Ok') #will not be printed if user gives a small string
print('loop terminated') #outside the loop
```

Lists

Lists

- A collection of data ordered and changeable. Allows duplicate members.
- Allows members of different types
- List members inside square brackets [], separated by commas
- Lists represent the concept of array
- Examples:

```
fruits= ["fig", "apple", "pear", 1, 2, 3]
2Dtable= [ [ 2, 3, 5] , [ 1, 4, 7 ]]
```

Lists: create, slices, copy, concat

```
>>> shoplist = ['apple', 'mango', 'carrot', 'banana', 'pear', 'fig']
>>> shoplist[0] #[0] is the first list item
'apple’
>>> shoplist[-1] # [-1] is the last list item
'fig'
>>> shoplist[0:2]
['apple', 'mango']
>>> shoplist[0:6:2] #from item 0 to 5, step 2
['apple', 'carrot', 'pear’]
>>> shoplist[6:2:-1] #from item 6 to 3, step -1
['fig', 'pear', 'banana']
>>> shoplist[-1] # 1 before end
‘fig'
>>> shoplist[1:-2] #from 1 to 2nd before end
['mango', 'carrot', 'banana’]
>>> new_list=shoplist[:] #make a copy of shoplist
>>> huge_list=new_list + shoplist #add two lists together
```

Loop through a list with `for` & `in`

```
>>> for each in shoplist:
   print(each)

apple
mango
carrot
banana
pear
```

```
>>> for each in shoplist:
       if each[1]=='a':
              print(each)


mango
carrot
banana
```

- Membership operator `in` can also be used to check whether specific item is in a list
- `each` is just a variable name that make code easy to read

Loop through 2D lists

```
# create a 2D list
a = [ [ 2, 3, 5] , [ 1, 4, 7 ] ]
print ("List before: a ={}".format(a))
# find list dimensions
rows = len(a)
cols = len(a[0])
# run the list and add 1 in every item
for row in range(rows):
    for col in range(cols):
        a[row][col] += 1
# print list items
print ("List after: a ={}".format(a))
```

List operations

List methods:

- `len(list)`: list length
- `max(list) & min(list)`: max and min values in list
- `list.reverse()`: reverse list (changes original)
- `list.append(x)`: add item *x* at the end of list
- `list.insert(i,x)`: insert item *x* at given position *i* in list
- `list.pop([i])`: remove item from the given position *i* or from the end of the list
- `list.index(x)`: return zero-based index of first item whose value is equal to *x*.
- `list.remove(x)`: remove first item whose value is equal to *x*
- `list.sort()`: sorts original list
- `sorted(list)`: sorts without changing original
- `list.count(x)`: return number of times *x* appears in list.

```
>>> shoplist
['apple', 'mango', 'carrot', 'banana', 'pear', 'fig']
>>> len(shoplist)
6
>>> shoplist.reverse()
>>> shoplist
['fig', 'pear', 'banana', 'carrot', 'mango', 'apple']
>>> shoplist.append('orange') # add at the end
>>> shoplist
['fig', 'pear', 'banana', 'carrot', 'mango', 'apple', 'orange']
>>> shoplist.insert(3,'grapes') # insert before position 3
>>> shoplist
['fig', 'pear', 'banana', 'grapes', 'carrot', 'mango', 'apple', 'orange']
>>> shoplist.pop(5) # delete item from index 5
'mango'
>>> shoplist
['fig', 'pear', 'banana', 'grapes', 'carrot', 'apple', 'orange']
```

`statistics` module

```
from statistics import *
```

`mean(list)` ->Arithmetic mean or average
`median(list)` ->Median
`mode(list)` ->Mode

`pstdev(list)` ->Standard deviation (population)

`pvariance(list)` ->Variance (population)

`stdev(list)` ->Standard deviation (sample)

`variance(list)` ->Variance (sample)

+ Basic functions
  - `max(list)`
  - `min(list)`
  - `sum(list)`

Tuples

- Sequence of immutable data, used as constants
- Tuples are like lists but **can’t** be modified
- Tuple members inside parentheses (), separated by commas, different types
- Same operations like lists
- Examples:

```
tup1 = ('physics', 'chemistry', 1997, 2000);
tup2 = () #empty tuple
tup3 = (50,) #single value tuple, should use one comma
```

Dictionaries

- Collection of data, unordered, changeable and indexed by *keys*.
- Keys can be integers, string or other objects and are unique
- Written with curly brackets {}, have keys and values.

```
bike =  {
   "brand": "Husqvarna",
   "model": "Silverpilen",
   "year": 1953
}
```

key

value

Dictionary operations

```
>>> bike =          {
  "brand": "Husqvarna",
  "model": "Silverpilen",
  "year": 1953
} # create a dictionary
>>> bike
{'brand': 'Husqvarna', 'model': 'Silverpilen', 'year': 1953}
>>> bike['model'] # Get the value of the model key
'Silverpilen’
>>> x = bike.get("brand") # get the value of brand key and assign it to x
>>> x
'Husqvarna'
>>> bike['year']=1955 #change value of year key
>>> bike
{'brand': 'Husqvarna', 'model': 'Silverpilen', 'year': 1955}
```

Dictionary: keys, values, items & copy

```
# use keys(), values() and items() in dictionary
# to print keys, values and key-value pairs
bike =  {
  "brand": "Husqvarna",
  "model": "Silverpilen",
  "year": 1953
}
print("Out-dict1: {}".format(bike.keys()))
print("Out-dict2: {}".format(bike.values()))
print("Out-dict3: {}".format(bike.items()))
new_bike=bike.copy() # copy an existing dictionary to a new variable
print("Out-dict4: {}".format(new_bike))
```

Loop through a Dictionary

```
bike =   {
  "brand": "Husqvarna",
  "model": "Silverpilen",
  "year": 1953
}
#printing keys
for key in bike:
    print("Out-dict5: {}".format(key))
#printing values
for key in bike:
    print("Out-dict6: {}".format(bike[key]))
#printing both keys and values
for key, value in bike.items():
        print("Out-dict7: {} {}".format(key,value))
```

Modify a dictionary

```
>>> bike =      {
  "brand": "Husqvarna",
  "model": "Silverpilen",
  "year": 1953
}
>>> bike["color"]='red’ # add a new key, value pair
>>> bike
{'brand': 'Husqvarna', 'model': 'Silverpilen', 'year': 1953, 'color':
'red'}
>>> bike.pop("color") #remove a key, if present, and return its value
'red'
>>> bike
{'brand': 'Husqvarna', 'model': 'Silverpilen', 'year': 1953}
>>> new_stuff={"year": [1955, 1960], "colors": ["black","white"]}
>>> new_stuff
{'year': [1955, 1960], 'colors': ['black', 'white']}
>>> bike.update(new_stuff) #merge a dictionary with another
>>> bike
{'brand': 'Husqvarna', 'model': 'Silverpilen', 'year': [1955, 1960],
'colors': ['black', 'white']}
```

Exercises

1. Write a python program that creates a list that contains the even numbers from 1 to 50. It will print the list and its length.
2. Write a python program that reads a string of words separated by the '?' character and assign them as items to a list. Then print each word one by one.
3. Write a python program that reads the number of positive numbers (or has a string with the positive number), which will then read and will put to a list. Subsequently it should:
    1. Print its minimum, maximum, and average
    2. Print a (vertical) histogram from characters '*'
    3. Print its cumulative list. the [1,2,3] has cumulative [1,3,6].
4. Write a python program that creates a list of strings that the user will type in the keyboard until the word 'end'. Next, it will sort then list alphanumerically ascending, print it, ask the user to give a string, which will be removed from the list (if any).
5. Write a python program that each element of the two-dimensional list a = [[2, 3, 5], [1, 4, 7]] will replace itself with its square.

ΠΑΝΕΠΙΣΤΗΜΙΟ
ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

Introduction to Information Systems and Applications

Course Unit 2: Data processing with python

M. Tzagarakis, V. Daskalou

School of Business Administration

Department of Economics

Functions and Exceptions

Functions

Use your own functions when you are writing the same snippet of code over and over again

definition -> `def function_name(parameter):`

`return` -> used to return the value

call: `function_name(parameter)`

```
#defines a functions that returns average of nums
def calc_avg(num_values):
        if len(num_values)>0:
                return sum(num_values)/len(num_values)
        else:
                return float('nan')
lista = [3,4,5,6,6,7,8,9]
print("Outfunc-1: Average: {0:.2f}".format(calc_avg(lista)))
```

Exceptions

- Robust code must handle errors and exceptions effectively, e.g. program does not throw and error with a division with zero
- Use try-exception block to catch and handle exceptions

```
#defines a functions that returns average of nums
def calc_avg(num_values):
                return sum(num_values)/len(num_values)
listb=[]
#handles division by zero exception
try:
    print("Outfunc-1: Average: {0:.2f}".format(calc_avg(listb)))
except ZeroDivisionError as detail:
    print("Outfunc-1: (Error): {}".format(float('nan')))
    print("Outfunc-1: (Error): {}".format(detail))
```

Files

Datasets in text files

We can use python to process open data in formats like:

- Text files: A text file contains human-readable characters and each line of text is terminated with a special character known as EOL (End of Line) character.
  - Text files use type .txt
- CSV (Comma Separated Value) text files: use “,” or “;” as delimiter between columns, or
  TSV (Tab Separated Value) text files: use tab as delimiter
  - CSV files use type .csv and TSV type .tsv

A user can read the contents of a text file or edit it using a text editor (such as MS notepad)

Read a text file (1)

```
with open("animals.txt", mode="r",encoding="utf-8") as my_file:
        for line in my_file:
            print(line)
```

Improvement: `print(line.rstrip("\n")) or print(line, end='')`

Content of file animals.txt

```
antelope
bear
cat
dog
elephant
fox
```

open(*file, mode='r', buffering=-1, encoding=None, errors=None, newline=None, closefd=True, opener=None*)

- `file`: file name
- `mode`:
  - **w** – open file for writing (deletes previous content)
  - **r** – open file for reading
  - **a** – open file for writing and append to existing content
- `encoding`: for text file by different countries we use UTF8

Read a text file (2)

- `myfile.read()` -> reads all file lines in one string
  `'antelope\nbear\ncat\ndog\nelephant\nfox'`
- `myfile.readlines()` -> reads all lines in a list
  `['antelope', 'bear', 'cat', 'dog', 'elephant', 'fox']`
- `myfile.read().splitlines()` -> reads all lines in a string and separates it in lines
  `['antelope\n', 'bear\n', 'cat\n', 'dog\n', 'elephant\n', 'fox']`
- `for line in myfile` -> reads each line one-by-one in the variable `line`

Read CSV files

Centent of file party_guests.csv

```
"name";"whose";"adults";"children"
"Anne";"Mum's";2;3
"Bod";"Dad's";1;1
"Susan";"Mine";2;2
"Marek";"Brother's";2;0
"Mairy";"Mum's";2;0
```

```
import csv
with open('party_guests.csv', mode='r', newline='',encoding='utf-8') as f:
        reader=csv.reader(f,delimiter=';',quotechar='"', quoting=csv.QUOTE_NONNUMERIC)
        header=next(reader)
        print(header)
        for row in reader:
                print(row)
```

1. Use csv module to manage CSV files
2. Opens csv with `open()`
3. Use `csv.reader()` to read from file. Define `delimiter`, `quotechar` and that non-quoted will be floats
4. Use `for in` to access each line one-by-one
5. Each row is a list and the value in each column in the list is a list member

Read from CSV file: find sums of columns

```
import csv
names=[]
whose_friends=[]
adults=[]
childrens=[]
with open('party_guests.csv', mode='r', newline='',encoding='utf-8',) as f:
    reader=csv.reader(f,delimiter=';',quotechar='"', quoting=csv.QUOTE_NONNUMERIC)
    header=next(reader)
    for row in reader:
          #members of list row are assigned to variables on the left
          #name,whose,nadult,nchildren=row
          name=row[0]
          whose=row[1]
          nadult=row[2]
          nchildren=row[3]
          names.append(name)
          whose_friends.append(whose)
          adults.append(nadult)
          childrens.append(nchildren)
print(sum(adults))
print(sum(childrens))
```

Write to a CSV file

Contents of output file myOutput.csv

```
"name";"whose";"adults";"children"
"Anne";"Mum's";2;3
"Bod";"Dad's";1;1
"Susan";"Mine";2;2
"Marek";"Brother's";2;0
"Mairy";"Mum's";2;0
```

```
import csv
data = [['name','whose','adults','children'],\
['Anne','Mum's',2.0,3.0],\
['Bod","Dad's",1,1]]
with open('myOutput.csv',mode='w', newline='') as myWFile:
    writer = csv.writer(myWFile, delimiter=';',quotechar='"',quoting=csv.QUOTE_NONNUMERIC)
    for line in data:
        writer.writerow(line)
```

1. Open output file with `open()` but with `mode=w` (write)
2. Use `csv.writer()` to write to output file. Define `delimiter`, `quotechar` and that non-quoted chars with be numeric
3. Use `for in` to access each list in output data list (variable `data` is a list of lists)
4. Use method `writerow()` of writer object to write a line in the output file with the specified format

Read from csv to dictionary (1)

```
# module cvs has functions to manage csv files
import csv
mguests=0
with open('party_guests.csv', mode='r', newline='',encoding='utf-8') as f:
    # reads lines in dictionary with keys the header names
    reader=csv.DictReader(f,delimiter=';',quotechar='"', quoting=csv.QUOTE_NONNUMERIC)
    # handles exceptions with try-except
    try:
        for row in reader:
            if row['whose']=="Mum's":
                print("Line: {!s}".format(row))
                mguests=mguests+row['adults']+row['children']
        print("Mum has {0} guests".format(mguests))
    except csv.Error as e:
            sys.exit('file %s, line %d: %s' % (filename, reader.line_num, e))
```

Read from csv to dictionary (2)

New issues:

- Use of `csv.DictReader:` Reads in an ordered dictionary each line:

```
OrderedDict([('name', 'Anne'), ('whose', 'Mum's'),
('adults', 2.0), ('children', 3.0)])
```

- Use `try-except` to handle errors

ΠΑΝΕΠΙΣΤΗΜΙΟ
ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

Introduction to Information Systems and Applications

Course Unit 2: Data processing with python

M. Tzagarakis, V. Daskalou

School of Business Administration

Department of Economics

Pandas

Why Pandas?

- Open-source python library for data analysis
- Include easy-to-use data structures and data analysis tools
- Used in a wide range of fields in academic and commercial domains, such as:
  - statistics,
  - analytics,
  - finance,
  - Economics
- Documentation:
  - https://pandas.pydata.org/pandas-docs/stable/index.html
  - 10 Minutes to pandas
  - Tutorials

Issues related to installation

- Better use pandas with Anaconda Python package
  - recall anaconda installation from Intro to Python
- Pandas library uses most of the functionalities of NumPy:
  - NumPy stands for Numerical Python
  - Is a library consisting of multidimensional array objects and a collection of routines for processing those arrays

```
In [1]: import pandas as pd

In [2]: import numpy as np

In [3]: import matplotlib.pyplot as plt
```

Data types

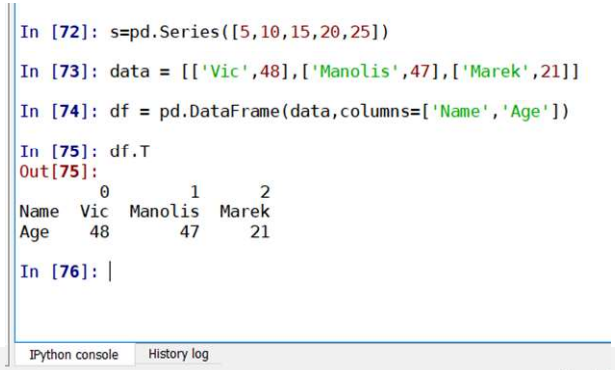
- Three (3) data types: Series, DataFrame, Panel
- Series:
  - 1 dimension
  - Homogeneous data, Size Immutable, Values of Data Mutable
- DataFrame:
  - 2 dimensions (tubular data: index or rows and columns)
  - Heterogeneous data, Size Mutable, Data Mutable
- Panel:
  - 3 dimensions
  - Heterogeneous data, Size Mutable, Data Mutable

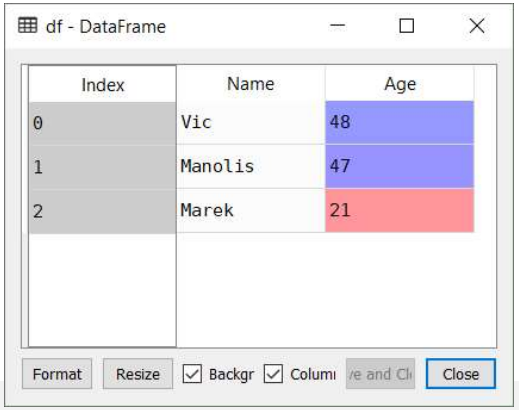
Examples of data types (1)

```
In [4]: s=pd.Series([5,10,15,20,25])
In [5]: s
Out[5]:
0     5
1    10
2    15
3    20
4    25
dtype: int64
```

Create a DataFrame from a list

```
In [6]: data = [['Vic',48],['Manolis',47],['Marek',21]]
In [7]: df = pd.DataFrame(data,columns=['Name','Age'])
In [8]: df
Out[8]:
Name Age
0 Vic 48
1 Manolis 47
2 Marek 21
```

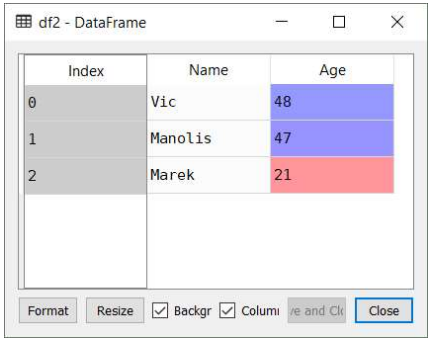




Examples of data types (2)

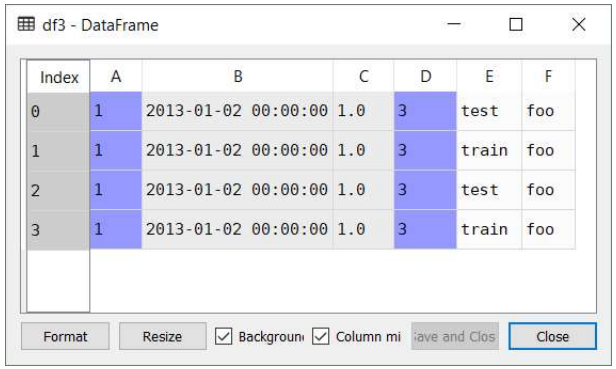
Create a DataFrame from a dictionary

```
In [11]: data2={'Name':['Vic','Manolis','Marek'],'Age':[48,47,21]}
In [12]: df2=pd.DataFrame(data=data2)
In [13]: df2
Out[13]:
Name Age
0 Vic 48
1 Manolis 47
2 Marek 21
```



Create a DataFrame of different column types

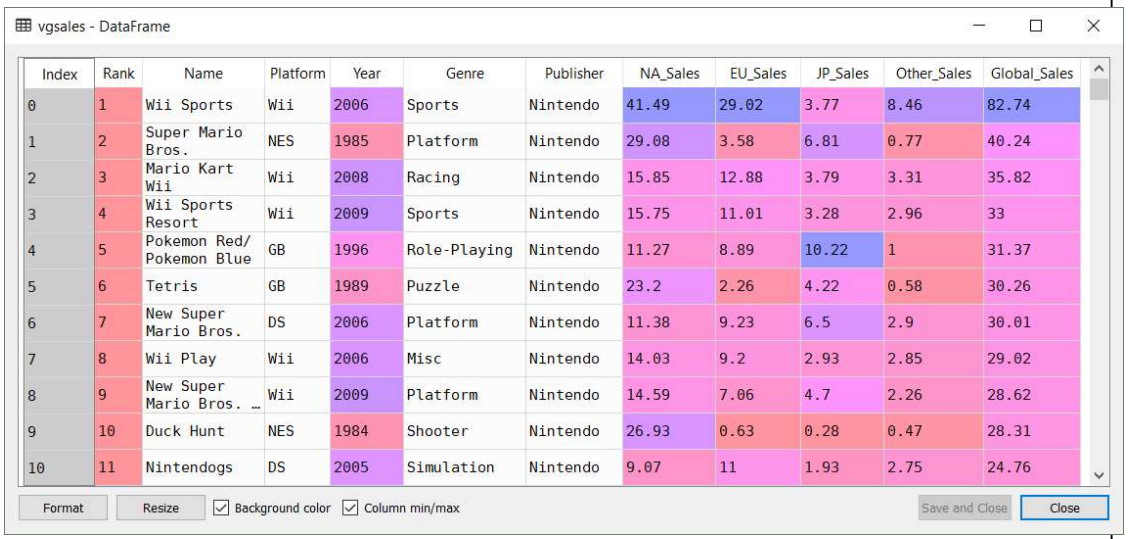
```
In [15]: df3 = pd.DataFrame({ 'A' : 1.,
    ...: 'B' : pd.Timestamp('20130102'),
    ...: 'C' : pd.Series(1,index=list(range(4)),dtype='float32'),
    ...: 'D' : np.array([3] * 4,dtype='int32'),
    ...: 'E' : pd.Categorical(["test","train","test","train"]),
    ...: 'F' : 'foo' })
In [16]: df3.dtypes
Out[16]:
A           float64
B    datetime64[ns]
C           float32
D             int32
E          category
F            object
dtype: object
```



More: https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html#pandas.DataFrame

Read from a CSV file

```
In [21]: vgsales=pd.read_csv('vgsales.csv')
In [22]: vgsales.dtypes
Out[22]:
Rank int64
Name object
Platform object
Year float64
Genre object
Publisher object
NA_Sales float64
EU_Sales float64
JP_Sales float64
Other_Sales float64
Global_Sales float64
dtype: object
```



Index	Rank	Name	Platform	Year	Genre	Publisher	NA_Sales	EU_Sales	JP_Sales	Other_Sales	Global_Sales
0	1	Wii Sports	Wii	2006	Sports	Nintendo	41.49	29.02	3.77	8.46	82.74
1	2	Super Mario Bros.	NES	1985	Platform	Nintendo	29.08	3.58	6.81	0.77	40.24
2	3	Mario Kart Wii	Wii	2008	Racing	Nintendo	15.85	12.88	3.79	3.31	35.82
3	4	Wii Sports Resort	Wii	2009	Sports	Nintendo	15.75	11.01	3.28	2.96	33
4	5	Pokemon Red/ Pokemon Blue	GB	1996	Role-Playing	Nintendo	11.27	8.89	10.22	1	31.37
5	6	Tetris	GB	1989	Puzzle	Nintendo	23.2	2.26	4.22	0.58	30.26
6	7	New Super Mario Bros.	DS	2006	Platform	Nintendo	11.38	9.23	6.5	2.9	30.01
7	8	Wii Play	Wii	2006	Misc	Nintendo	14.03	9.2	2.93	2.85	29.02
8	9	New Super Mario Bros. …	Wii	2009	Platform	Nintendo	14.59	7.06	4.7	2.26	28.62
9	10	Duck Hunt	NES	1984	Shooter	Nintendo	26.93	0.63	0.28	0.47	28.31
10	11	Nintendogs	DS	2005	Simulation	Nintendo	9.07	11	1.93	2.75	24.76

View top and bottom rows of frame

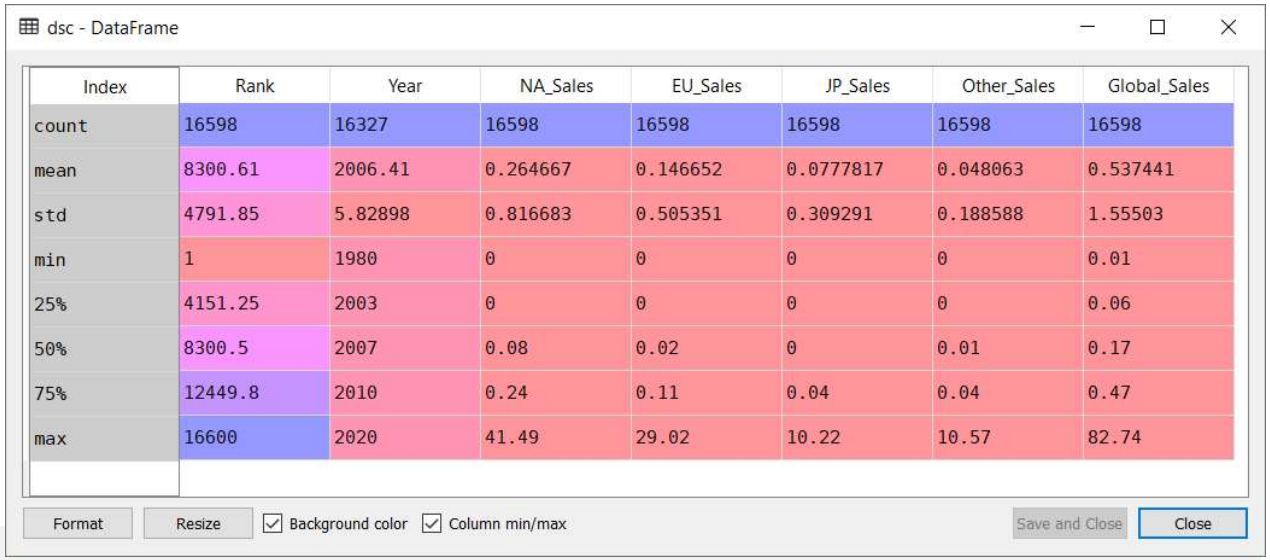
```
In [23]: vgsales.head()
Out[23]:
   Rank                      Name      ...       Other_Sales  Global_Sales
0     1                Wii Sports      ...              8.46         82.74
1     2         Super Mario Bros.      ...              0.77         40.24
2     3            Mario Kart Wii      ...              3.31         35.82
3     4         Wii Sports Resort      ...              2.96         33.00
4     5  Pokemon Red/Pokemon Blue      ...              1.00         31.37

[5 rows x 11 columns]
In [24]: vgsales.tail(5)
Out[24]:
        Rank     ...      Global_Sales
16593  16596     ...              0.01
16594  16597     ...              0.01
16595  16598     ...              0.01
16596  16599     ...              0.01
16597  16600     ...              0.01

[5 rows x 11 columns]
```

Index, columns, and underlying data

```
In [25]: vgsales.index [check if returns a range object]
Out[25]: RangeIndex(start=0, stop=16598, step=1)

In [26]: vgsales.columns
Out[26]:
Index(['Rank', 'Name', 'Platform', 'Year', 'Genre', 'Publisher', 'NA_Sales',
       'EU_Sales', 'JP_Sales', 'Other_Sales', 'Global_Sales'],
      dtype='object')
In [32]: vgsales.values
Out[32]:
array([[1, 'Wii Sports', 'Wii', ..., 3.77, 8.46, 82.74],
       [2, 'Super Mario Bros.', 'NES', ..., 6.81, 0.77, 40.24],
       [3, 'Mario Kart Wii', 'Wii', ..., 3.79, 3.31, 35.82],
       ...,
       [16598, 'SCORE International Baja 1000: The Official Game', 'PS2',
        ..., 0.0, 0.0, 0.01],
       [16599, 'Know How 2', 'DS', ..., 0.0, 0.0, 0.01],
       [16600, 'Spirits & Spells', 'GBA', ..., 0.0, 0.0, 0.01]],
      dtype=object)
```

Descriptive statistics

```
In [29]: dsc=vgsales.describe()

In [30]: dsc
    ...:
Out[30]:
                Rank          Year      ...        Other_Sales  Global_Sales
count  16598.000000  16327.000000      ...       16598.000000  16598.000000
mean    8300.605254   2006.406443      ...           0.048063      0.537441
std     4791.853933      5.828981      ...           0.188588      1.555028
min        1.000000   1980.000000      ...           0.000000      0.010000
25%     4151.250000   2003.000000      ...           0.000000      0.060000
50%     8300.500000   2007.000000      ...           0.010000      0.170000
75%    12449.750000   2010.000000      ...           0.040000      0.470000
max    16600.000000   2020.000000      ...          10.570000     82.740000

[8 rows x 7 columns]
```



Index	Rank	Year	NA_Sales	EU_Sales	JP_Sales	Other_Sales	Global_Sales
count	16598	16327	16598	16598	16598	16598	16598
mean	8300.61	2006.41	0.264667	0.146652	0.0777817	0.048063	0.537441
std	4791.85	5.82898	0.816683	0.505351	0.309291	0.188588	1.55503
min	1	1980	0	0	0	0	0.01
25%	4151.25	2003	0	0	0	0	0.06
50%	8300.5	2007	0.08	0.02	0	0.01	0.17
75%	12449.8	2010	0.24	0.11	0.04	0.04	0.47
max	16600	2020	41.49	29.02	10.22	10.57	82.74

Getting data

Getting a column

```
In [11]: vgsales['Name']
Out[11]:
0 Wii Sports
1 Super Mario Bros.
2 Mario Kart Wii
3 Wii Sports Resort
4 Pokemon Red/Pokemon Blue
5 Tetris
6 New Super Mario Bros.
7 Wii Play
8 New Super Mario Bros. Wii
9 Duck Hunt
10 Nintendogs
11 Mario Kart DS
12 Pokemon Gold/Pokemon Silver

...
16596 Know How 2
16597 Spirits & Spells
Name: Name, Length: 16598, dtype: object
```

Getting rows with slices

```
In [12]: vgsales[3:5]
Out[12]:
   Rank                      Name  ...  Other_Sales  Global_Sales
3     4          Wii Sports Resort  ...         2.96         33.00
4     5  Pokemon Red/Pokemon Blue  ...         1.00         31.37

[2 rows x 11 columns]
```

Select data by label: loc

```
# extract a subset of dataframe. Specify slices for rows and columns.
# more about DataFrame.loc https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.loc.html
print(vgsales.loc[0:5,'Name':'Year'])
#loc with only label (row) returns a series object
x=vgsales.loc[0]
print(x)
#loc with a list of labels for rows returns a dataframe
y=vgsales.loc[[0,5]]
print(y)
#loc with single label for row and column returns a cell, the Name of row 0
z=vgsales.loc[0,'Name']
#Slice with labels for row and single label for column. Return a Series object
a=vgsales.loc[3:7,'Name']
# loc specific columns for all rows
b=vgsales.loc[:,['Name','Global_Sales']]
#conditional
c=vgsales.loc[vgsales['Rank']<5]
z=vgsales.loc[((vgsales['Rank']>=1) & (vgsales['Rank']<=100))]
#convert dataframe and use to_string() to print it as a whole
print(z.to_string())
#Using the isin() method for filtering
d=vgsales[vgsales['Platform'].isin(['Wii','DS','PS4’])]
```

Selection by position: iloc (1/2)

Return the row in a specific position

```
In [4]: vgsales.iloc[5]
Out[4]:
Rank                    6
Name               Tetris
Platform               GB
Year                 1989
Genre              Puzzle
Publisher        Nintendo
NA_Sales             23.2
EU_Sales             2.26
JP_Sales             4.22
Other_Sales          0.58
Global_Sales        30.26
Name: 5, dtype: object
```

Specify slices with integers

```
In [6]: vgsales.iloc[0:3,0:3]
Out[6]:
Rank Name Platform
0 1 Wii Sports Wii
1 2 Super Mario Bros. NES
2 3 Mario Kart Wii Wii
```

Selection by position: iloc (2/2)

Specify lists of specific integer position locations (can specify also only rows or columns)

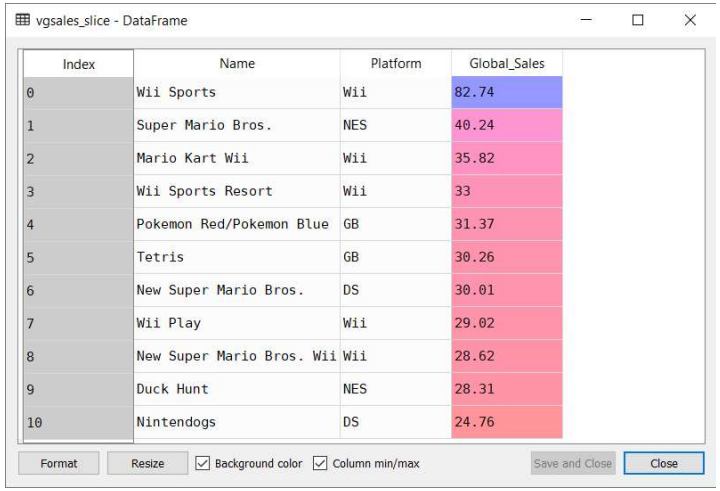
```
In [7]: vgsales.iloc[[1,2,4],[0,2]]
Out[7]:
   Rank Platform
1     2      NES
2     3      Wii
4     5       GB
```

Specify an exact cell position

```
In [8]: vgsales.iloc[1][1]

Out[8]: 'Super Mario Bros.'
```

Sorting

```
In [111]: vgsales_slice.sort_values(by='Name')
Out[111]:
                          Name Platform  Global_Sales
9                    Duck Hunt      NES         28.31
2                Mario Kart Wii      Wii         35.82
6        New Super Mario Bros.       DS         30.01
8    New Super Mario Bros. Wii      Wii         28.62
10                   Nintendogs       DS         24.76
4     Pokemon Red/Pokemon Blue       GB         31.37
1            Super Mario Bros.      NES         40.24
5                      Tetris       GB         30.26
7                    Wii Play      Wii         29.02
0                  Wii Sports      Wii         82.74
3            Wii Sports Resort      Wii         33.00
```



Setting

Create a DataFrame from a list of list

```
In [12]:
data=pd.DataFrame([[1,2,3],[4,5,6]],
columns=['A','B','C'])

In [13]: data
    ...:
Out[13]:
A B C
0 1 2 3
1 4 5 6
```

Create a new column in DataFrame (with value 1)

```
In [14]: data['E']=1

In [15]: data
Out[15]:
A B C E
0 1 2 3 1
1 4 5 6 1
```

Set value to a column (value 2 to column E)

```
In [16]: data.loc[:,'E']=2
In [17]: data
Out[17]:
A B C E
0 1 2 3 2
1 4 5 6 2
```

Set a value to a specific cell (value 10 to cell [0,0])

```
In [18]: data.iloc[0,0]=10
In [19]: data
Out[19]:
A B C E
0 10 2 3 2
1 4 5 6 2
```

Set values based on conditions (set 0 to cells with value >2)

```
In [20]: data[data>2]=0
In [21]: data
Out[21]:
A B C E
0 0 2 0 2
1 0 0 0 2
```

Operations

- Statistics
- Apply
- Aggregate
- Histogramming
- Grouping
- Pivot tables

Operations: Statistics

Create a DataFrame from a list of list (3 rows, 3 columns)

```
data=pd.DataFrame([[1,2,3],[4,5,6],[7,8,9]],
columns=['A','B','C'])
data
Out[24]:
   A  B  C
0  1  2  3
1  4  5  6
2  7  8  9
```

Calculate mean of values in requested axis (by default axis 0 is index)

```
data.mean()
Out[25]:
A    4.0
B    5.0
C    6.0
dtype: float64
```

Calculate mean of values in requested axis (1 is by rows)

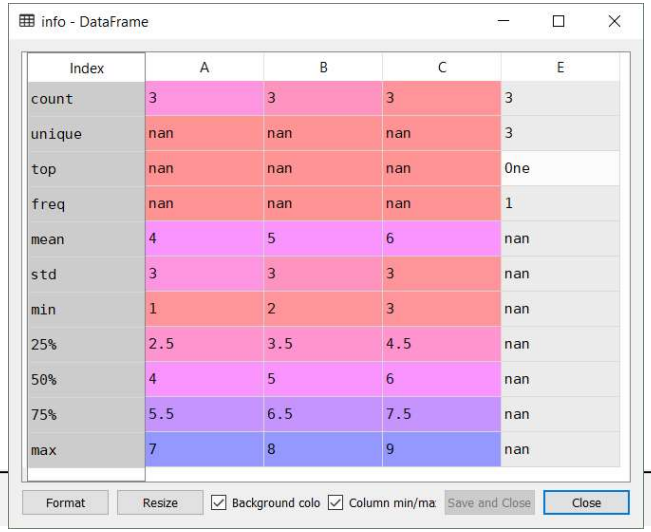
```
data.mean(1)
Out[26]:
0    2.0
1    5.0
2    8.0
dtype: float64
```

Calculate sum of values in requested axis (by default axis 0 is index)

```
In [27]: data.sum()
Out[27]:
A    12
B    15
C    18
dtype: int64
```

Calculate descriptive statistics to all type of columns

```
In [30]: data['E']=['One','Two','Three']
In [31]: data
Out[31]:
A B C E
0 1 2 3 One
1 4 5 6 Two
2 7 8 9 Three
In [32]: info=data.describe(include='all')
```



Operations: apply

applymap() applies a function to every single element in the entire dataframe.

More about apply functions: https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.apply.html

```
import pandas as pd
# create a function called add10
def add10(x):
    # that, if x is a string,
    if type(x) is str:
        # just returns it untouched
        return x
    # but, if not, return it by adding 10
    elif x:
        return x+10
    # and leave everything else
    else:
        return
#create a dataframe
data=pd.DataFrame([[1,2,3],[4,5,6],[7,8,9]], columns=['A','B','C’])
#apply function add10 to each element of dataframe data
data10=data.applymap(add10)
```

Operations: aggregate

Aggregate using one or more operations over the specified axis. agg is an alias for aggregate. Use the alias.

```
In [43]: data
Out[43]:
   A  B  C
0  1  2  3
1  4  5  6
2  7  8  9
```

Aggregate these functions over the rows.

```
In [44]: data.agg(['sum', 'min'])
Out[44]:
      A   B   C
sum  12  15  18
min   1   2   3
```

Aggregate over the columns.

```
In [45]: data.agg("mean", axis="columns")
Out[45]:
0    2.0
1    5.0
2    8.0
dtype: float64
```

Different aggregations per column.

```
In [46]: data.agg({'A' : ['sum', 'min'], 'B' : ['min', 'max']})
Out[46]:
        A    B
max   NaN  8.0
min   1.0  2.0
sum  12.0  NaN
```

Operations: Histogramming

Series.value_counts(): Returns object containing counts of unique values.

```
In [55]: data
Out[55]:
   A  B  C    E
0  7  2  3  Foo
1  4  5  6  Bar
2  7  8  9  Foo

In [56]: data['A'].value_counts()
Out[56]:
7    2
4    1
Name: A, dtype: int64

In [57]: data['E'].value_counts()
Out[57]:
Foo    2
Bar    1
Name: E, dtype: int64
```

Grouping

Get a slice of vgsales DataFrame from row 0 to 10 and the columns Name, Platform and Global_Sales

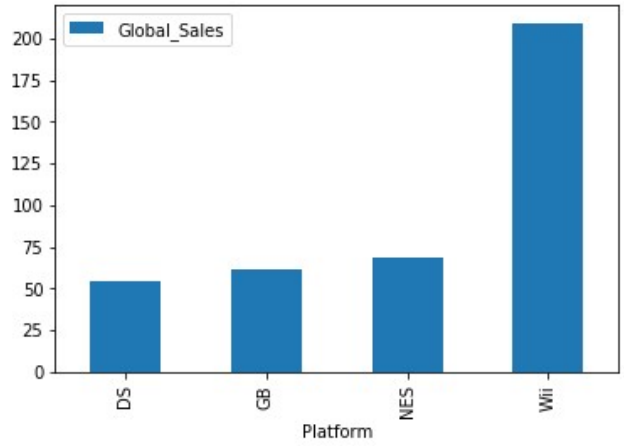
```
In [64]: vgsales_slice=vgsales.loc[0:10,['Name','Platform','Global_Sales']]
In [65]: vgsales_slice
Out[65]:
                         Name Platform  Global_Sales
0                  Wii Sports      Wii         82.74
1           Super Mario Bros.      NES         40.24
2              Mario Kart Wii      Wii         35.82
3           Wii Sports Resort      Wii         33.00
4    Pokemon Red/Pokemon Blue       GB         31.37
5                      Tetris       GB         30.26
6       New Super Mario Bros.       DS         30.01
7                    Wii Play      Wii         29.02
8   New Super Mario Bros. Wii      Wii         28.62
9                   Duck Hunt      NES         28.31
10                 Nintendogs       DS         24.76
```

Grouping and then applying the sum() function to the resulting groups.

```
In [66]: vgsales_slice.groupby('Platform').sum()
Out[66]:
Platform    Global_Sales
DS                 54.77
GB                 61.63
NES                68.55
Wii               209.20
```

Pivot tables (1/2)

```
In [120]: pv=vgsales_slice.pivot_table(index='Platform',aggfunc='sum')
In [124]: pv
Out[124]:
          Global_Sales
Platform
DS               54.77
GB               61.63
NES              68.55
Wii             209.20
In [125]: pv.plot(kind='bar')
Out[125]: <matplotlib.axes._subplots.AxesSubplot at 0x1be4d569630>
```



Pivot tables (1/2)

```
vgsales_slice2=vgsales.loc[0:20]

In [139]: pv=vgsales_slice2.pivot_table('NA_Sales',index='Publisher',
columns='Genre',aggfunc='sum',margins=True)
```

pv - DataFrame

Index	Action	Misc	Platform	Puzzle	Racing	Role-Playing	Shooter	Simulation	Sports	All
Microsoft Game Studios	nan	14.97	nan	nan	nan	nan	nan	nan	nan	14.97
Nintendo	nan	18.78	67.83	23.2	25.66	26.69	26.93	9.07	75.27	273.43
Take-Two Interactive	16.44	nan	nan	nan	nan	nan	nan	nan	nan	16.44
All	16.44	33.75	67.83	23.2	25.66	26.69	26.93	9.07	75.27	304.84

Format Resize Background color Column min/max Save and Close Close

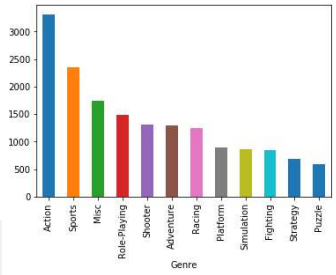
Categoricals

Pandas can include categorical data in a DataFrame.

```
In [144]:
vgsales['Platform']=vgsales['Platform'].astype('category')
In [145]:
vgsales['Year']=vgsales['Year'].astype('category')
In [146]:
vgsales['Genre']=vgsales['Genre'].astype('category')
In [147]: vgsales.dtypes
Out[147]:
Rank              int64
Name             object
Platform       category
Year           category
Genre          category
Publisher        object
NA_Sales        float64
EU_Sales        float64
JP_Sales        float64
Other_Sales     float64
Global_Sales    float64
Dtype: object
```

Grouping by a categorical column and then apply the size() function can give us a frequency table of a categorical column (histogram)

```
In [154]:
vgsales.groupby('Genre').size().sort_values(ascending=Fa
lse)
Out[154]:
Genre
Action          3316
Sports          2346
Misc            1739
Role-Playing    1488
Shooter         1310
Adventure       1286
Racing          1249
Platform         886
Simulation       867
Fighting         848
Strategy         681
Puzzle           582
dtype: int64
In [155]:
vgsales_by_genre=vgsales.groupby('Genre').size().sort_va
lues(ascending=False)
In [156]: vgsales_by_genre.plot(kind='bar')
Out[156]: <matplotlib.axes._subplots.AxesSubplot at
0x1be4d68b5c0>
```

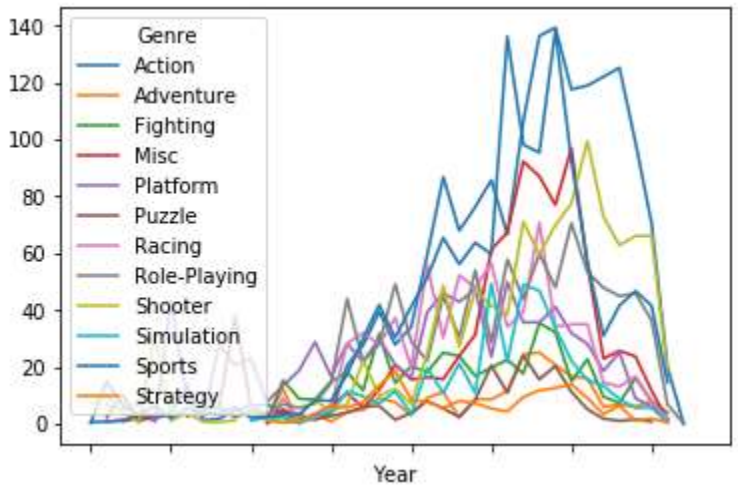


Plotting

https://pandas.pydata.org/pandas-docs/stable/visualization.html#visualization

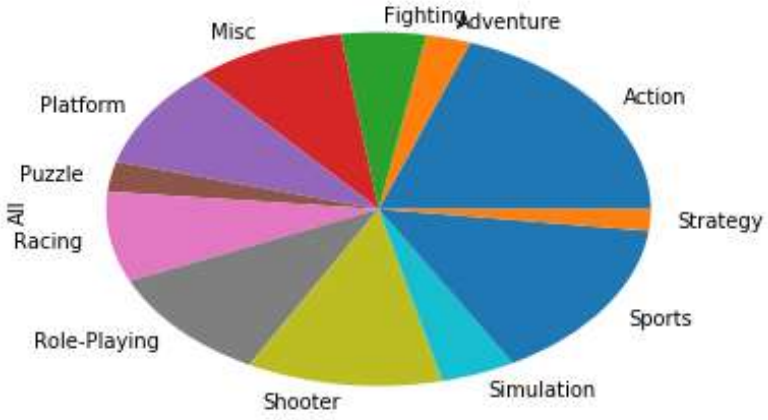
Create a pivot table of Global sales per Genre for every year and plot it

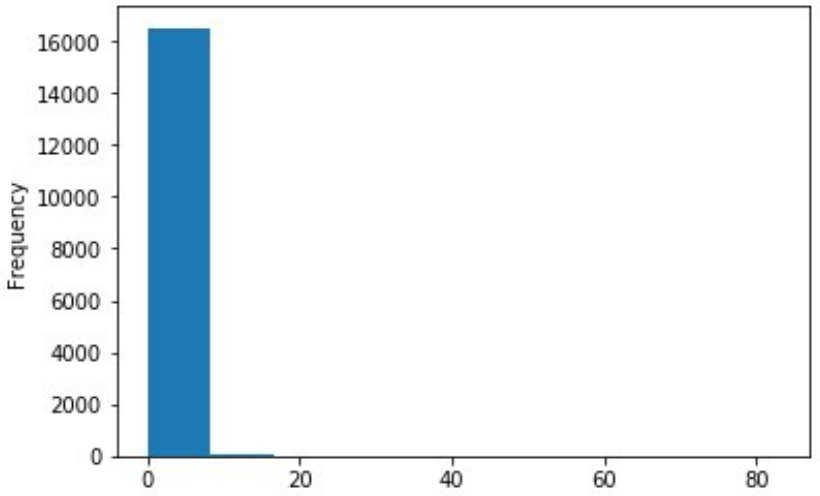
```
vgsales.pivot_table('Global_Sales', index='Year',
columns='Genre', aggfunc='sum').plot()
```



Plotting

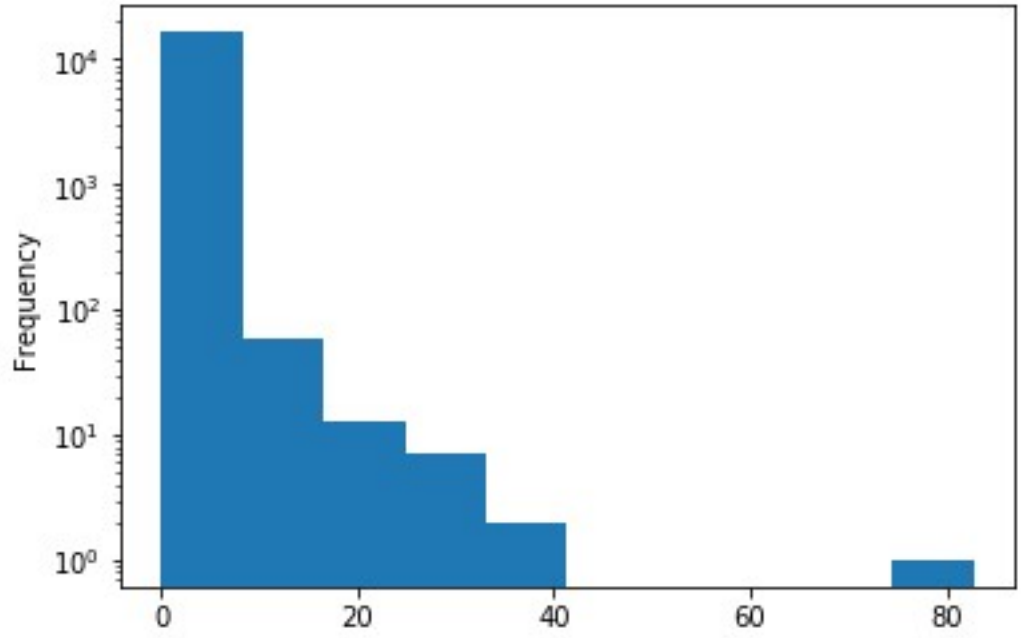
```
pv=vgsales.pivot_table('Global_Sales', index='Year',
columns='Genre', aggfunc='sum',margins='all')

plt.figure()

pv.loc['All','Action':'Strategy'].plot.pie()

plt.figure()

vgsales['Global_Sales'].plot.hist()
```





Plotting

```
vgsales['Global_Sales'].plot.hist(logy=True)
```



Notes

Note on History of Published Version

The present work is the edition 1.0

Reference note

References

1. Severance, C. R., Blumenberg, S., & Hauser, E. (2016). *Python for Everybody: Exploring Data in Python 3*. CreateSpace Independent Publishing Platform. Under CC BY-NC-SA retrieved on Oct. 2018 by https://open.umn.edu/opentextbooks/textbooks/python-for-everybody-exploring-data-using-python-3
2. VanderPlas, J. (2016). *Python data science handbook: essential tools for working with data*. Under CC-BY-NC-ND license retrieved on Nov. 2018 by https://jakevdp.github.io/PythonDataScienceHandbook/
3. Python Schools, http://www.pythonschool.net/
4. Non-Programmer's Tutorial for Python 3, http://en.wikibooks.org/wiki/Non-Programmer%27s_Tutorial_for_Python_3/Print_version
5. Python Programming http://en.wikibooks.org/wiki/Python_Programming
6. The Python Tutorial, https://docs.python.org/3/tutorial/index.html
7. Pandas Documentation, https://pandas.pydata.org/pandas-docs/stable/

License Notes

Preservation Notices

Any reproduction or adaptation of the material should include:

- the Reference Note,
- the Licensing Note,
- the declaration of Notices Preservation,
- the Use of Third Parties Work Note (if available),

together with the accompanied URLs.

Note of use of third parties work(1/2

This work makes use of the following works:

Pictures/Shapes/Charts/Photos:

Image 1: Guido van Rossum, By Doc Searls (2006oscon_203.JPG) [CC BY-SA 2.0 (http://creativecommons.org/licenses/by-sa/2.0)], via Wikimedia Commons, Source: https://en.wikipedia.org/wiki/Guido_van_Rossum

Image 2: Top programming languages, By TIOBE Software B.V. [CC BY-SA 4.0 (http://creativecommons.org/licenses/by-sa/4.0)], via Wikimedia Commons, Source: https://commons.wikimedia.org/wiki/File:Tiobeindex.png