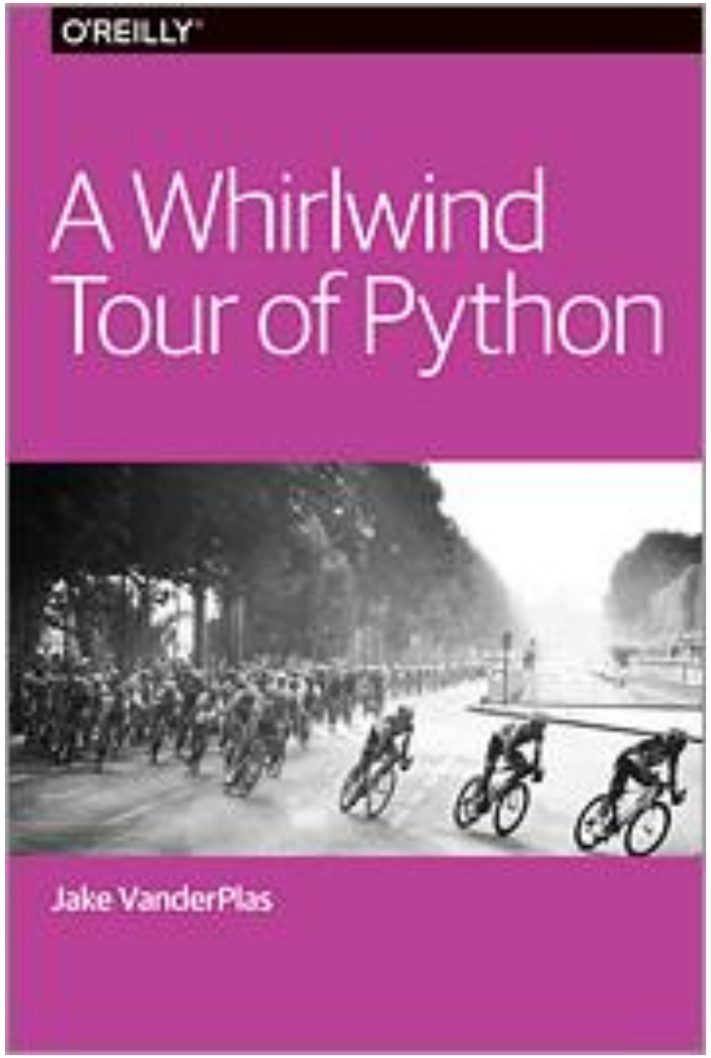
mirror_mod.use_x = False
mirror_mod.use_y = True
mirror_mod.use_z = False
elif _operation == "MIRROR_Z":
mirror_mod.use_x = False
mirror_mod.use_y = False
mirror_mod.use_z = True
#selection at the end -add back the deselected mirror modifier object
mirror_ob.select= 1
modifier_ob.select=1
bpy.context.scene.objects.active = modifier_ob
print("Selected" + str(modifier_ob)) # modifier ob is the active ob
#mirror_ob.select = 0

PYTHON for Data Science

M. Hamdi Özçelik

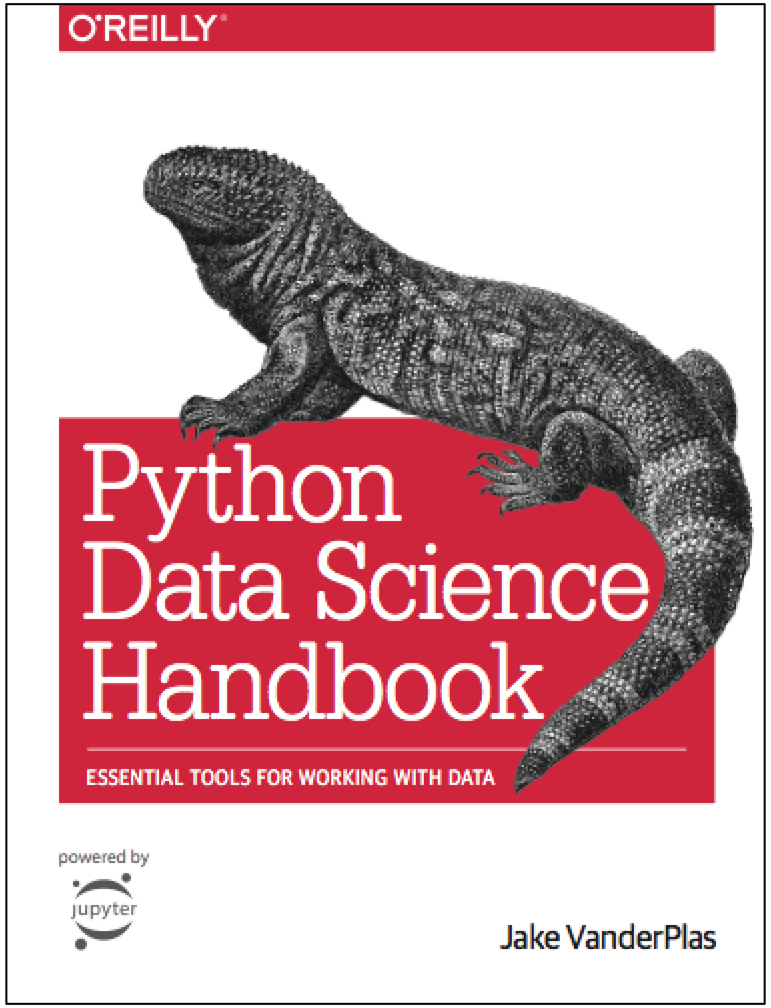
11.10.2021



https://jakevdp.github.io/WhirlwindTourOfPython/index.html

"Any fool can know. The point is to understand."

—Albert Einstein



O'REILLY

Python Data Science Handbook

ESSENTIAL TOOLS FOR WORKING WITH DATA

powered by

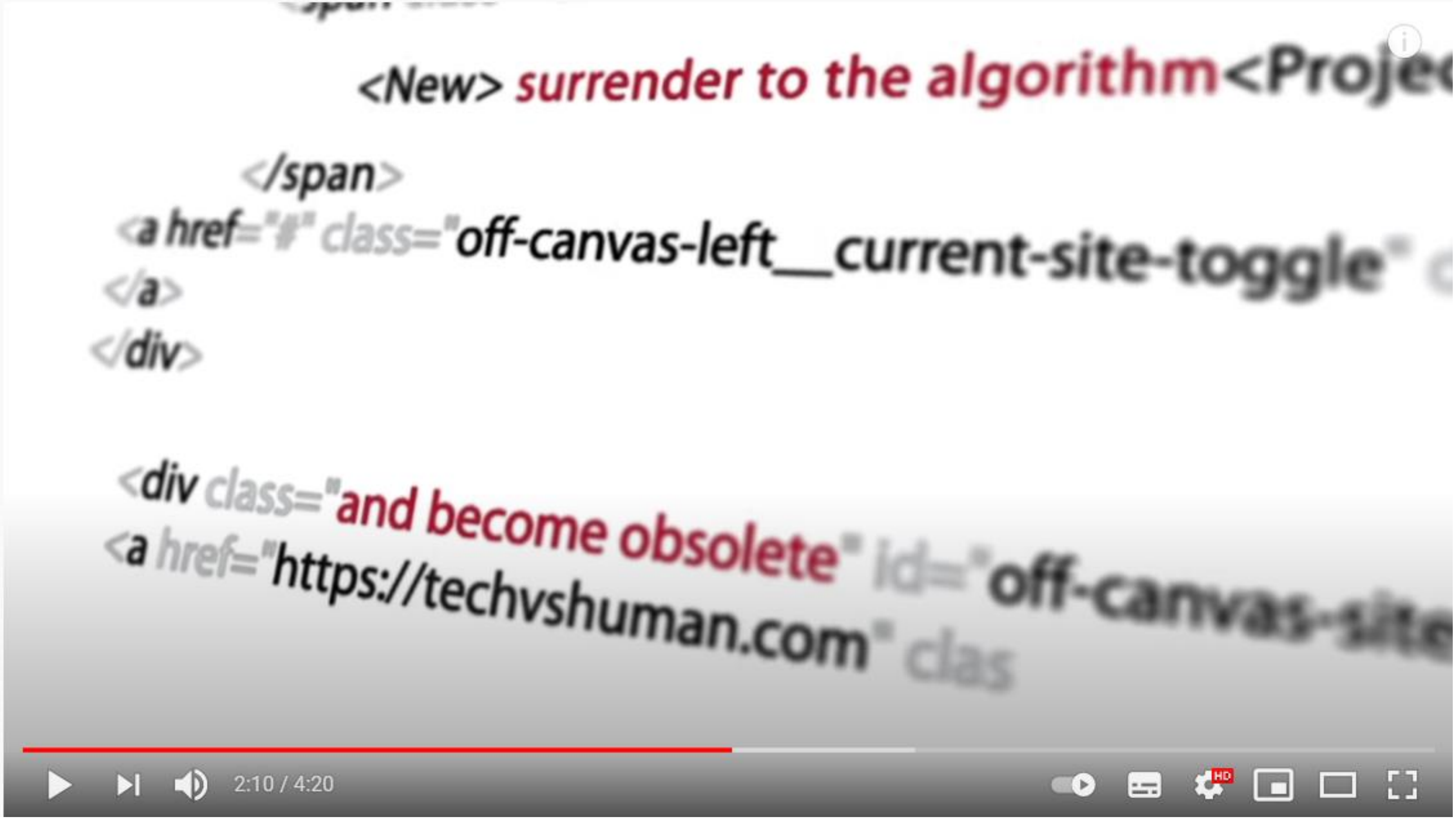
jupyter

Jake VanderPlas

https://jakevdp.github.io/PythonDataScienceHandbook/

"Anyone who has never made a mistake has never tried anything new."

—Albert Einstein



```
<span class
        <New> surrender to the algorithm<Proje
    </span>
  <a href="#" class="off-canvas-left__current-site-toggle" c
  </a>
 </div>

  <div class="and become obsolete" id="off-canvas-site
  <a href="https://techvshuman.com" clas
```

https://www.youtube.com/watch?v=XUVS5d3-Bis

PROGRAMMING LANGUAGES

easy to learn,
easy to use & maintain

The Fourth Generation Languages (4GL)
High level languages designed for a specific purpose
Another software reads the code and processes

```
def MySum (a,b):
    return a+b
```

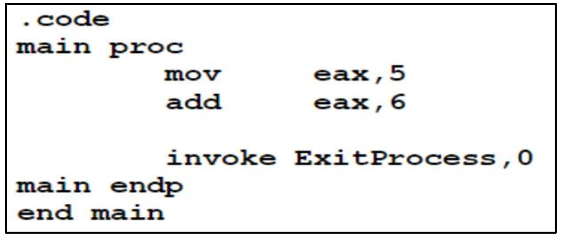
Python, SQL, R, ABAP, ...

The Third Generation Languages (3GL)
General purpose procedural languages
A compiler / interpreter converts the code to assembly code or byte code

```
int sum(int a, int b)
{
  return (a+b);
}
```

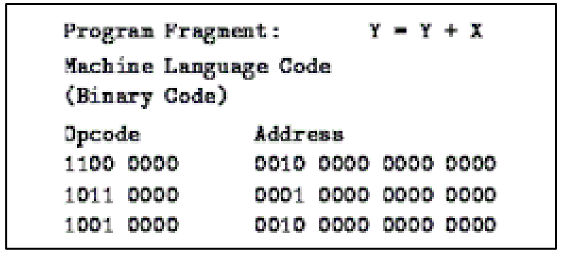
Python, C, C++, C#, Basic, Java, Prolog..

The Second Generation Languages (2GL)
“The assembly language”
An assembler converts that languages code to machine code



```
.code
main proc
        mov     eax,5
        add     eax,6

        invoke ExitProcess,0
main endp
end main
```

The First Generation Programming languages (1GL)
“The machine language”
Specific to CPU, instructions given to CPU directly



```
Program Fragment:       Y = Y + X
Machine Language Code
(Binary Code)
Opcode          Address
1100 0000       0010 0000 0000 0000
1011 0000       0001 0000 0000 0000
1001 0000       0010 0000 0000 0000
```

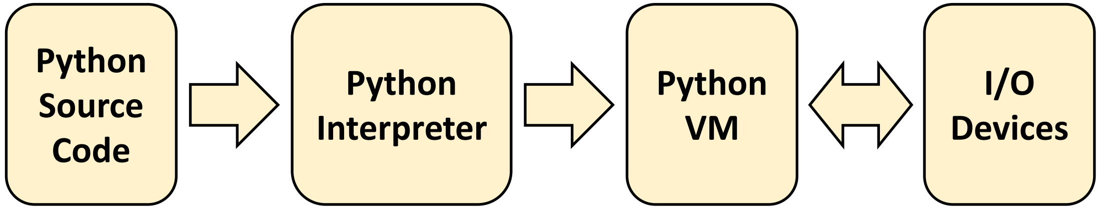
fast and flexible

PYTHON LANGUAGE

Guido van Rossum

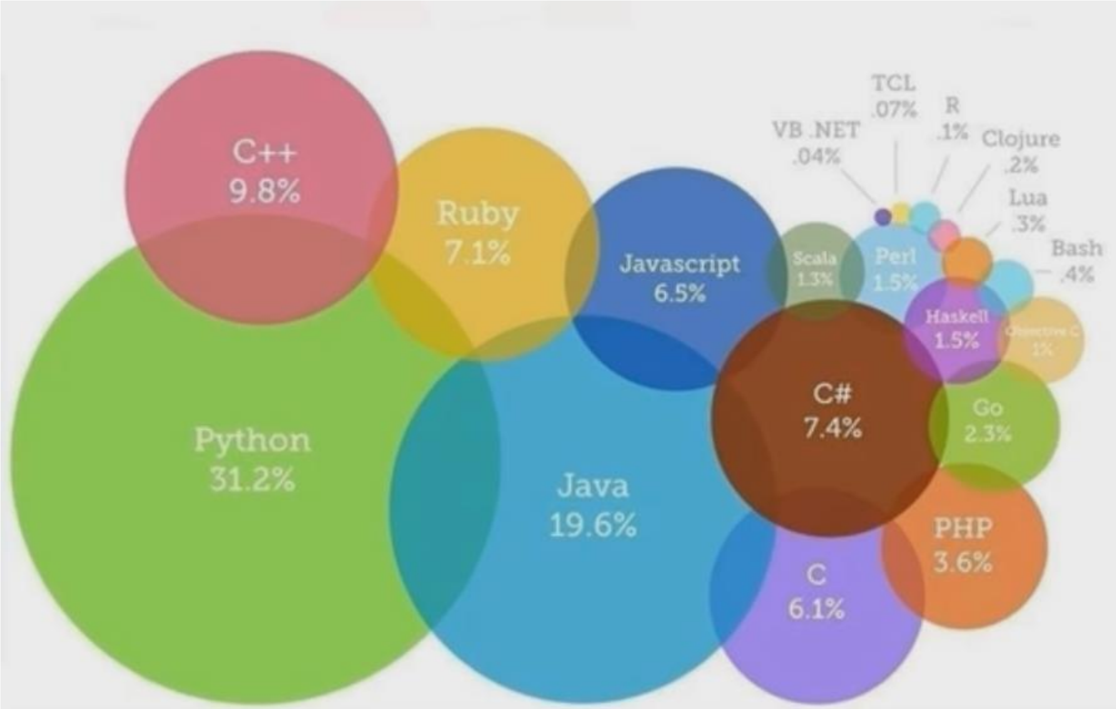
the creator of the Python programming language

- Python is a general-purpose programming language.
- Python is an interpreted language. To run a code written in Python you need a Python interpreter. The interpreter converts the Python source code into bytecode and then that bytecode is executed by the Python virtual machine.
- Python is an object-oriented programming language.
- IDE: Integrated Development Environment



Python Source Code → Python Interpreter → Python VM ⇔ I/O Devices

file names as ".py"

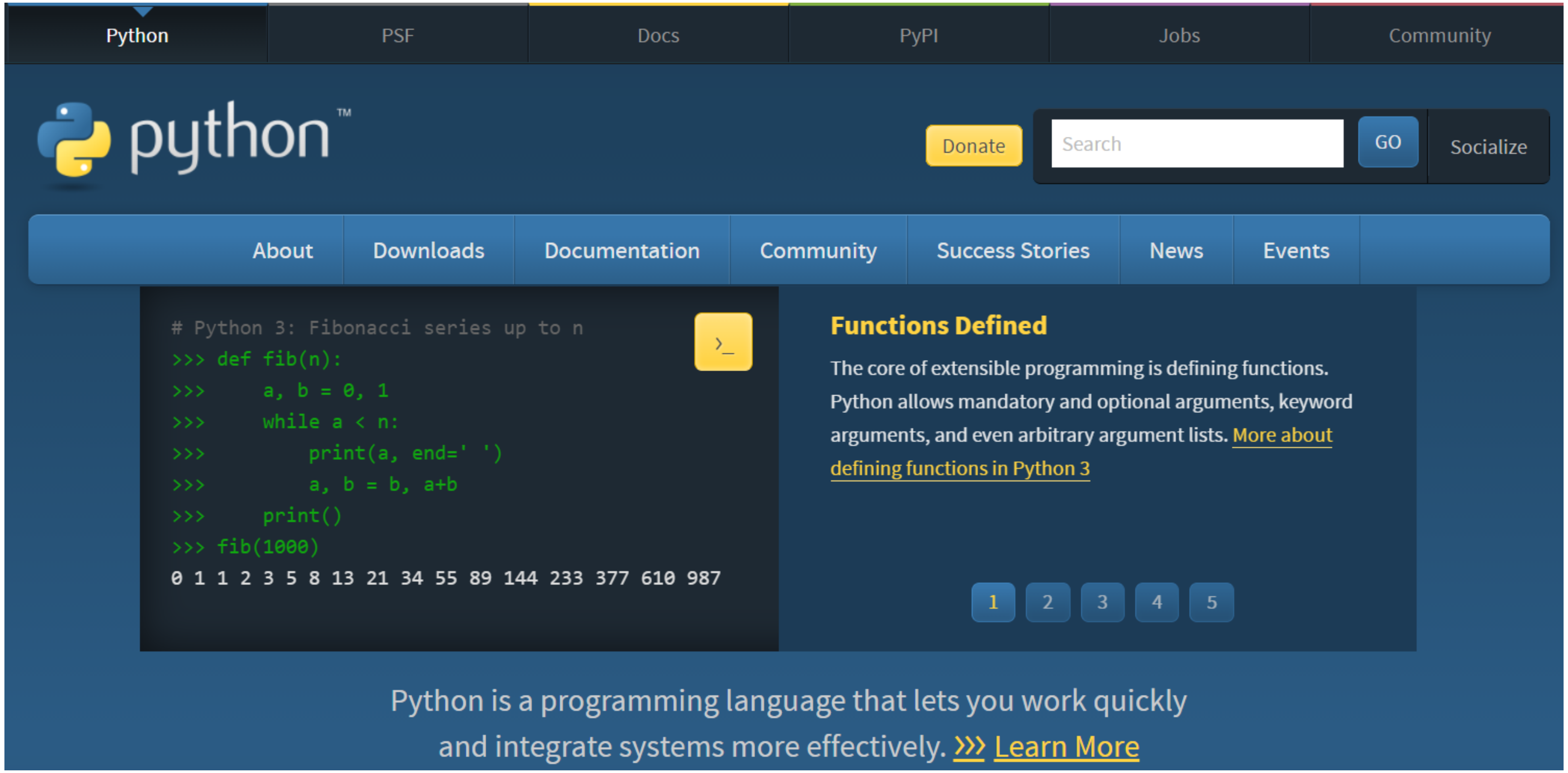


PYTHON LANGUAGE

Zen of Python

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Readability counts.

https://en.wikipedia.org/wiki/Zen_of_Python



```
# Python 3: Fibonacci series up to n
>>> def fib(n):
>>>     a, b = 0, 1
>>>     while a < n:
>>>         print(a, end=' ')
>>>         a, b = b, a+b
>>>     print()
>>> fib(1000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
```

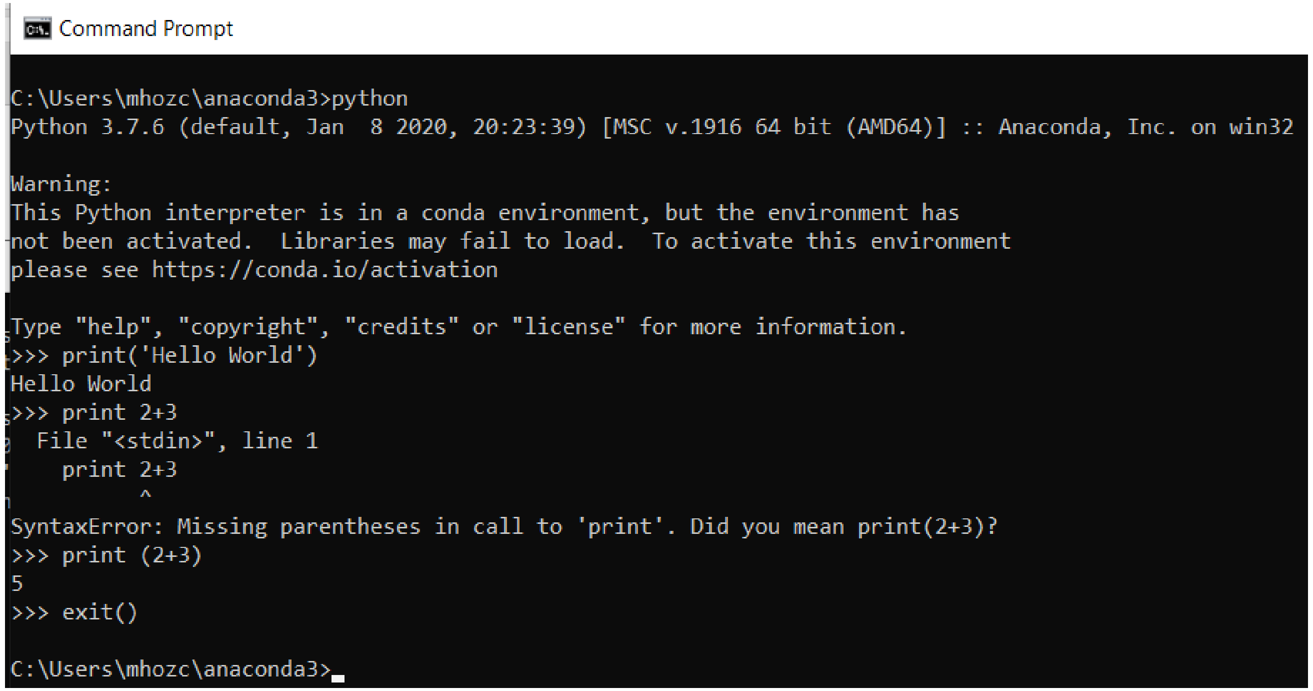
Functions Defined

The core of extensible programming is defining functions. Python allows mandatory and optional arguments, keyword arguments, and even arbitrary argument lists. More about defining functions in Python 3

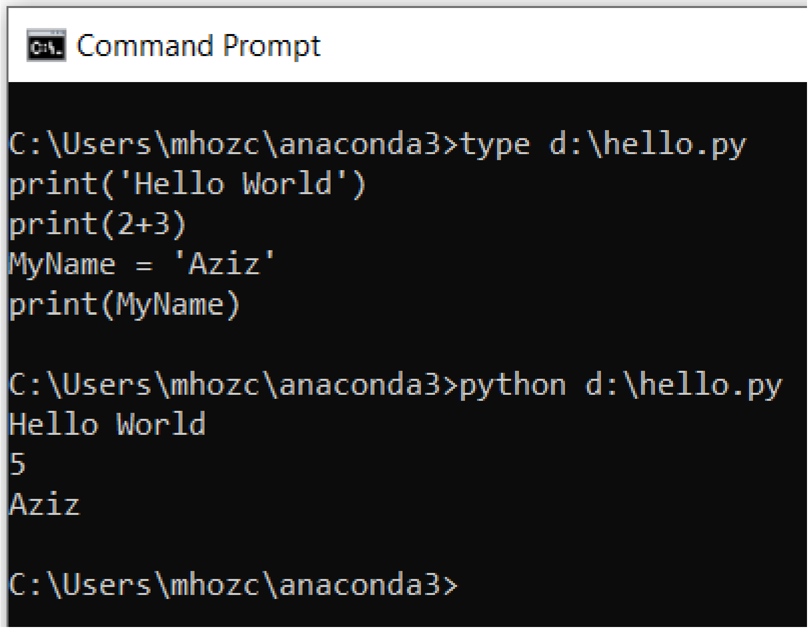
Python is a programming language that lets you work quickly and integrate systems more effectively. >>> Learn More

https://www.python.org/

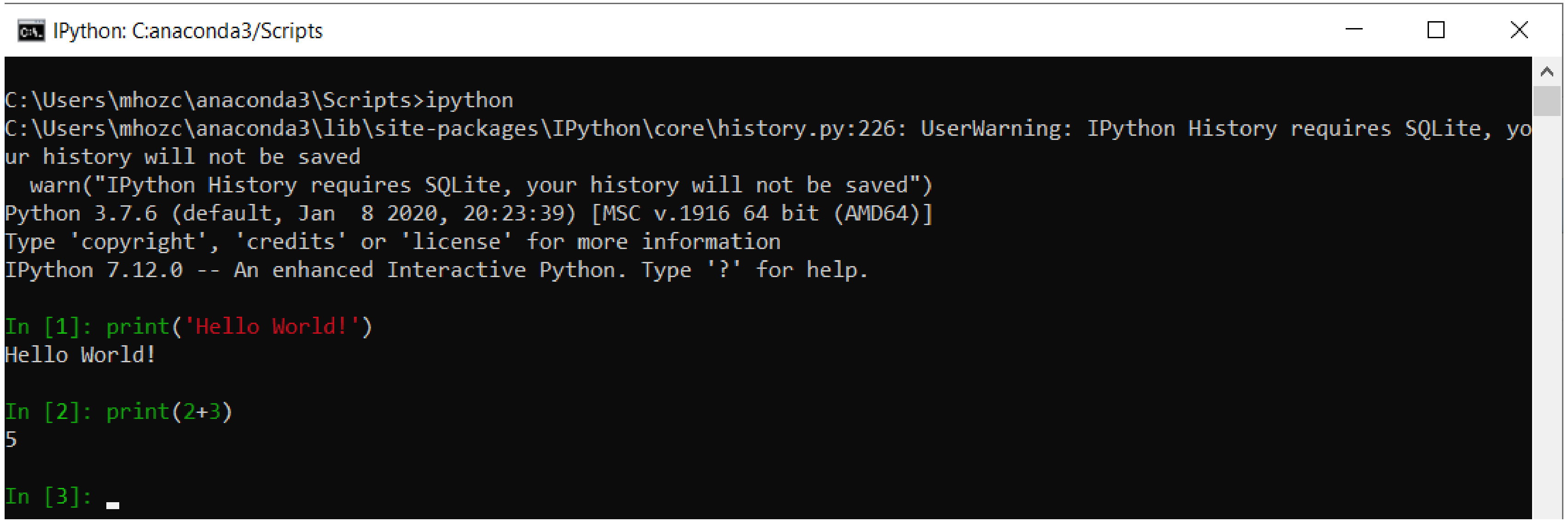
PYTHON SHELL



```
C:\Users\mhozc\anaconda3>python
Python 3.7.6 (default, Jan  8 2020, 20:23:39) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32

Warning:
This Python interpreter is in a conda environment, but the environment has
not been activated.  Libraries may fail to load.  To activate this environment
please see https://conda.io/activation

Type "help", "copyright", "credits" or "license" for more information.
>>> print('Hello World')
Hello World
>>> print 2+3
  File "<stdin>", line 1
    print 2+3
          ^
SyntaxError: Missing parentheses in call to 'print'. Did you mean print(2+3)?
>>> print (2+3)
5
>>> exit()

C:\Users\mhozc\anaconda3>
```



```
C:\Users\mhozc\anaconda3>type d:\hello.py
print('Hello World')
print(2+3)
MyName = 'Aziz'
print(MyName)

C:\Users\mhozc\anaconda3>python d:\hello.py
Hello World
5
Aziz

C:\Users\mhozc\anaconda3>
```



```
C:\Users\mhozc\anaconda3\Scripts>ipython
C:\Users\mhozc\anaconda3\lib\site-packages\IPython\core\history.py:226: UserWarning: IPython History requires SQLite, yo
ur history will not be saved
  warn("IPython History requires SQLite, your history will not be saved")
Python 3.7.6 (default, Jan  8 2020, 20:23:39) [MSC v.1916 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 7.12.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: print('Hello World!')
Hello World!

In [2]: print(2+3)
5

In [3]:
```

WHY PYTHON?

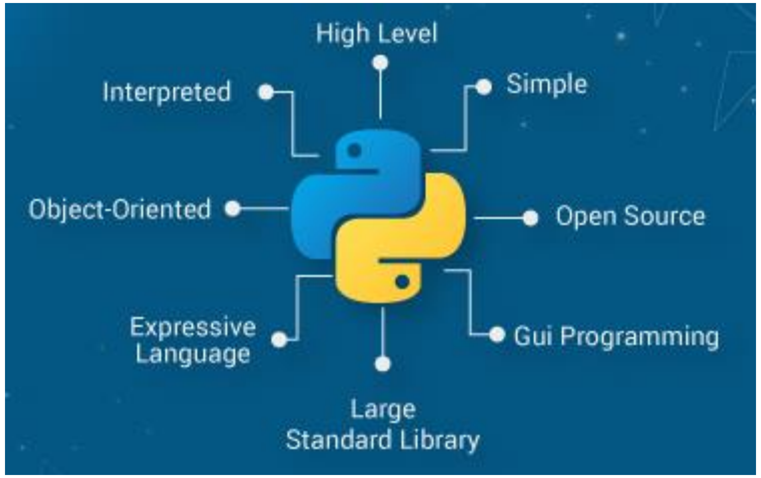
https://www.python.org/doc/essays/blurb/

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

Why Python?

(Cons of Python?)

1. Free
2. Easy to learn
3. Highly productive
4. Runs on (almost) any operating system
5. Widely used by the data science community
6. Widely used in academics
7. Rich set of libraries
8. Easy to automate

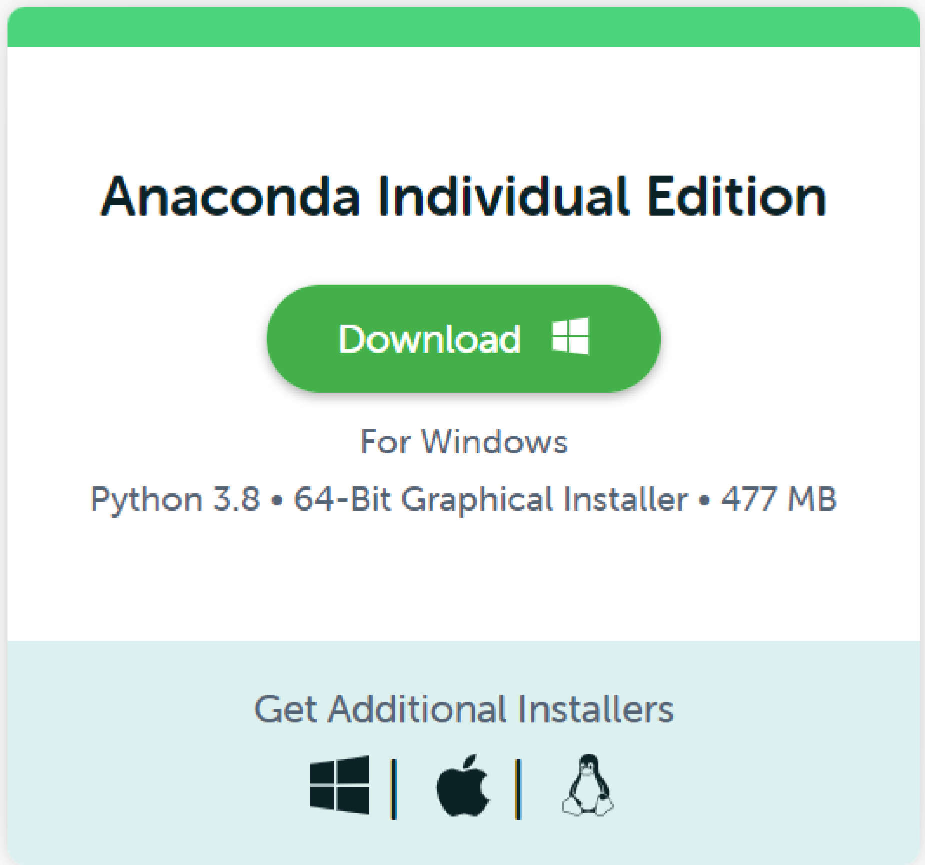


Many libraries

- **Numpy**: support for large, multi-dimensional arrays and matrices
- **Pandas**: data manipulation and analysis
- the generation of Microsoft Excel files using the **Python Excel library**,
- graphics libraries such as **Matplotlib** and PyOpenGL,
- machine learning using libraries such as **Scikit-learn** (SKLearn) and TensorFlow.
- web frameworks such as **Django**/Flask,
- email clients such as smtplib and imaplib,
- …

JUPYTER NOTEBOOK

ANACONDA.



Anaconda Individual Edition

For Windows

Python 3.8 • 64-Bit Graphical Installer • 477 MB

Get Additional Installers

jupyter

Project Jupyter exists to develop open-source software, open-standards, and services for interactive computing across dozens of programming languages.

Project Jupyter is a non-profit, open-source project, born out of the IPython Project in 2014 as it evolved to support interactive data science and scientific computing across all programming languages. Jupyter will always be 100% open-source software, free for all to use and released under the liberal terms of the modified BSD license.
Jupyter is developed in the open on GitHub, through the consensus of the Jupyter community. For more information on our governance approach, please see our Governance Document.

https://jupyter.org/about

PYTHON LANGUAGE

- **Python is a case-sensitive language**

It means a variable named as X and a variable named x are two different variables

- **Multiple lines** are allowed for a single statement, **white spaces** has no effect (except indentation)

```
print('Hello',
      'World')
```

- **Comments** starts with the character #

```
# Hello World example
print('Hello World')
```

- The NoneType is a variable type and it has a single value, **None**.

This is used to represent nothingness (i.e., null values).

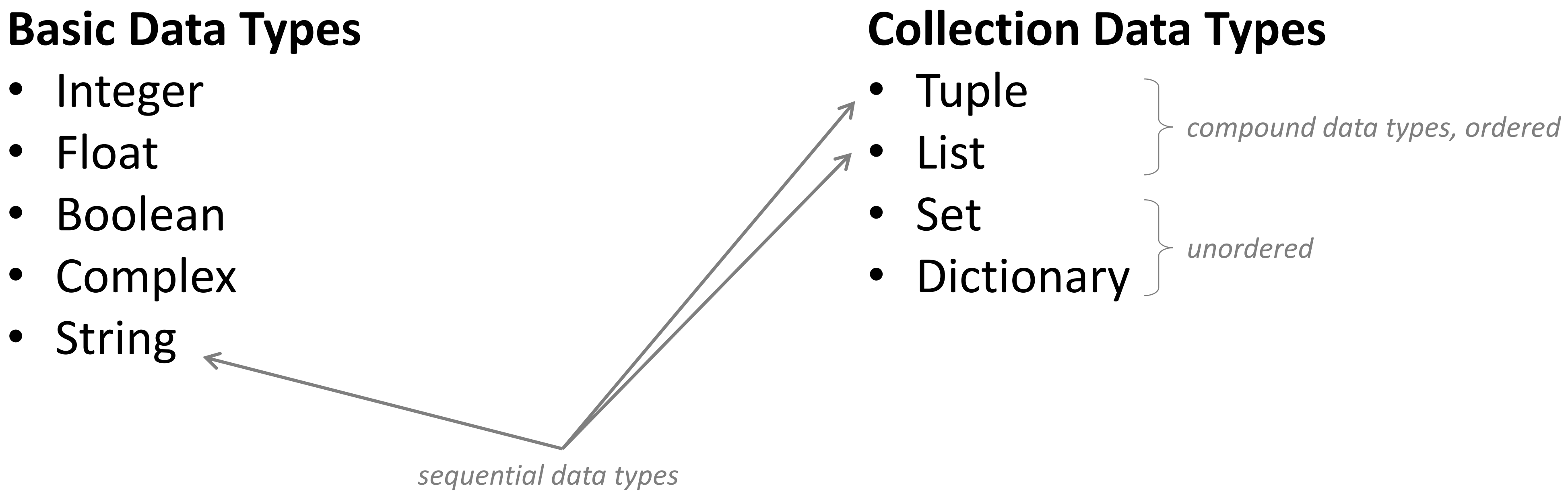
- Help() function is used to get help

```
help(len)
```

- Tab-completion of object contents are available

```
# first write “pri” and then press TAB
```

DATA TYPES



Basic Data Types

- Integer
- Float
- Boolean
- Complex
- String

Collection Data Types

- Tuple
- List
- Set
- Dictionary

compound data types, ordered

unordered

sequential data types

- An **tuple** is defined by items that are separated by commas and should be enclosed with "(" and ")". Items with different data types could form a tuple.

- The items in a **list** are separated by commas and should be enclosed with "[" and "]"

- The items in a **set** are separated by commas and should be enclosed with "{" and "}". No duplicates.

- A **dictionary** item is a key-value pair in the form of {key1: value1}. The keys are unique.

LANGUAGE BASICS

CONVERTING (CASTING) DATA TYPES

```
float(2)       2.0

int(2.3)       2

str(2.4)       '2.4'
```

ESCAPE CHARACTERS

\t	tab
\n	newline
\’	a single quote character
\"	a double quote character
\\	backslash character

```
print('hello world’)

print('hello\nworld’)
print(r'hello\nworld')
```

COMPUTATION ORDER

```
print(2*3+4)   10

print(2+3*4)   14
```

ASSIGNMENT vs COMPARISON

```
x = 3

(x == 3)
(x != 3)
```

LANGUAGE BASICS

TYPE OF ERRORS

1. **Syntactic** error: about the syntax
2. **Semantic** error: about the logic

QUOTATION CHARACTERS

```
x = "İSTANBUL"
```

```
x = 'İSTANBUL'
```

ELEMENTS IN A SEQUENCE

C	U	S	T	O	M	E	R
0	1	2	3	4	5	6	7
-8	-7	-6	-5	-4	-3	-2	-1

Negative indexing

STRING AS A SEQUENCE OF CHARACTERS

up to 2

```
x = "CUSTOMER"

X[0:2]          # CU
X[0:7]          # CUSTOME
X[2:5]          # STO

X[-1]           # R
X[::2]          # CSOE
```

LANGUAGE BASICS

MUTABLE / IMMUTABLE

1. **Mutable** means we can change the value(s).
2. **Immutable** means we can not change the value(s), i.e. read-only. Tuples are immutable. The keys at dictionaries are immutable.

ALIASING / CLONING

ALIASING: Objects referencing to the same data. Hence if data is changed both changed (similar to pointers in C).

CLONING: Objects referencing to different memory areas for their own data. Since their data are independent of each other, if one of them is changed the others' value is not effected

NESTED DATA TYPES

Lists and tuples allow nesting of lists and tuples

```
a = ['Customer', 'Analytics', ['SQL', 'Python', 'R Language'], 2021]
a.index('Analytics')
a[2][2][0]
```

INTEGER OPERATORS

Operator	Returns	Comments
x + y	int	Returns the sum of x and y
x − y	int	Returns the difference of x and y
x*y	int	Returns the product of x and y
x/y	float	Returns the quotient of x divided by y
x//y	int	Returns the integer quotient of x divided by y
x% y	int	Returns x modulo y. This is the remainder of dividing x by y
−x	int	Returns the negation of x
x&y	int	Returns the bit-wise and of x and y
x | y	int	Returns the bit-wise or of x and y
x ˆ y	int	Returns the bit-wise exclusive or of x and y
x << y	int	Returns a bit-wise shift left of x by y bits. Shifting left by 1 bit multiplies x by 2
x >> y	int	Returns a bit-wise right shift of x by y bits
˜ x	int	Returns an integer where each bit in the x has been inverted. x+ x = −1 for all x
abs(x)	int	Returns the absolute value of x
divmod(x, y)	(q,r)	Returns the quotient q and the remainder r as a tuple
float(x)	float	Returns the float representation of x
hex(x)	str	Returns a hexadecimal representation of x as a string
int(x)	int	Returns x
oct(x)	str	Return an octal representation of x as a string
pow(x, y[, z])	int	Returns x to the y power modulo z. If z is not specified then it returns x to the y power
repr(x)	str	Returns a string representation of x
str(x)	str	Returns a string representation of x

FLOAT OPERATORS

Operator	Returns	Comments
x + y	float	Returns the sum of x and y
x – y	float	Returns the difference of x and y
x*y	float	Returns the product of x and y
x/y	float	Returns the quotient of x divided by y
x//y	float	Returns the quotient of integer division of x divided by y. However, the result is still a float
x% y	float	Returns x modulo y. This is the remainder of dividing x by y
abs(x)	int	Returns the absolute value of x
divmod(x, y)	(q,r)	Returns the quotient q and the remainder r as a tuple. Both q and r are floats, but integer division is performed. The value r is the whole and fractional part of any remainder. The value q is a whole number
float(x)	float	Returns the float representation of x
int(x)	int	Returns the floor of x as an integer
pow(x, y)	float	Returns x to the y power
repr(x)	str	Returns a string representation of x
str(x)	str	Returns a string representation of x

STRING OPERATORS

Operator	Returns	Comments
s+t	str	Return a new string which is the concatenation of s and t
s in t	bool	Returns True if s is a substring of t and False otherwise
s==t	bool	Returns True if s and t refer to strings with the same sequence of characters
s>=t	bool	Returns True if s is lexicographically greater than or equal to t
s<=t	bool	Returns True if s is lexicographically less than or equal to t
s>t	bool	Returns True if s is lexicographically greater than t
s<t	bool	Returns True if s is lexicographically less than t
s!=t	bool	Returns True if s is lexicographically not equal to t
s[i]	str	Returns the character at index i in the string. If i is negative then it returns the character at index len(s)−i
s[[i]:[j]]	str	Returns the slice of characters starting at index i and extending to index j−1 in the string. If i is omitted then the slice begins at index 0. If j is omitted then the slice extends to the end of the list. If i is negative then it returns the slice starting at index len(s)+i (and likewise for the slice ending at j)
s ∗ i	str	Returns a new string with s repeated i times
i ∗ s	str	Returns a new string with s repeated i times
chr(i)	str	Return the ASCII character equivalent of the integer i
float(s)	int	Returns the float contained in the string s
int(s)	int	Returns the integer contained in the string s
len(s)	int	Returns the number of characters in s
ord(s)	int	Returns the ASCII decimal equivalent of the single character string s
repr(s)		Returns a string representation of s. This adds an extra pair of quotes to s
str(s)	str	Returns a string representation of s. In this case you get just the string s

STRING METHODS

Method	Returns	Comments
s.capitalize()	str	Returns a copy of the string s with the first character upper case
s.endswith(suffix[, start[, end]])	bool	Returns True if s ends with the specified suffix, False otherwise. With optional start, test s beginning at that position. With optional end, stop comparing s at that position. suffix can also be a tuple of strings to try
s.find(sub[, start[, end]])	int	Returns the lowest index in s where substring sub is found, such that sub is contained within s[start:end]. Optional arguments start and end are interpreted as in slice notation. Return −1 on failure
s.index(sub[, start[, end]])	int	Like s.find() but raise ValueError when the substring is not found
s.isalnum()	bool	Returns True if all characters in s are alphanumeric and there is at least one character in s, False otherwise
s.isalpha()	bool	Returns True if all characters in s are alphabetic and there is at least one character in s, False otherwise
s.isdecimal()	bool	Returns True if there are only decimal characters in s, False otherwise
s.isdigit()	bool	Returns True if all characters in s are digits and there is at least one character in s, False otherwise
s.isidentifier()	bool	Returns True if s is a valid identifier according to the language definition
s.islower()	bool	Returns True if all cased characters in s are lowercase and there is at least one cased character in s, False otherwise
s.isnumeric()	bool	Returns True if there are only numeric characters in s, False otherwise
s.isprintable()	bool	Returns True if all characters in s are considered printable in repr() or s is empty, False otherwise
s.isspace()	bool	Returns True if all characters in s are whitespace and there is at least one character in s, False otherwise
s.istitle()	bool	Returns True if s is a titlecased string and there is at least one character in s, i.e. upper- and titlecase characters may only follow uncased characters and lowercase characters only cased ones. Return False otherwise
s.isupper()	bool	Returns True if all cased characters in s are uppercase and there is at least one cased character in s, False otherwise
s.join(sequence)	str	Returns a string which is the concatenation of the strings in the sequence. The separator between elements is s
s.lower()	str	Returns a copy of the string s converted to lowercase
s.partition(sep)	(h,sep,t)	Searches for the separator sep in s, and returns the part before it, the separator itself, and the part after it. If the separator is not found, returns s and two empty strings
s.replace (old, new[, count])	str	Returns a copy of s with all occurrences of substring old replaced by new. If the optional argument count is given, only the first count occurrences are replaced
s.split([sep[, maxsplit]])	string list	Returns a list of the words in s, using sep as the delimiter string. If maxsplit is given, at most maxsplit splits are done. If sep is not specified or is None, any whitespace string is a separator and empty strings are removed from the result
s.splitlines([keepends])	string list	Returns a list of the lines in s, breaking at line boundaries. Line breaks are not included in the resulting list unless keepends is given and true
s.startswith (prefix[, start[, end]])	bool	Returns True if s starts with the specified prefix, False otherwise. With optional start, test s beginning at that position. With optional end, stop comparing s at that position. Prefix can also be a tuple of strings to try
s.strip([chars])	str	Returns a copy of the string s with leading and trailing whitespace removed. If chars is given and not None, removes characters in chars instead.
s.title()	str	Returns a titlecased version of s, i.e. words start with title case characters, all remaining cased characters have lower case
s.upper()	str	Returns a copy of s converted to uppercase
s.zfill(width)	str	Pad a numeric string s with zeros on the left, to fill a field of the specified width. The string s is never truncated

TUPLE OPERATORS and METHODS

Operator	Returns	Comments
t.count()	int	
t.index()	element	
t1 + t2	tuple	Returns a new tuple which is the concatenation of t1 and t2
tuple(s)	tuple	converts to a tuple
t1 * n		Duplicates the tuple n times
len(t)		Returns the number of elements in the tuple
min(t)	float	Returns the minimum value of elements in the tuple
max(t)	float	Returns the maximum value of elements in the tuple
sum(t)	float	Returns the sum of the element values in the tuple
any(t)	bool	Returns TRUE if any one of the elements in the tuple is TRUE, otherwise FALSE
all(t)	bool	Returns TRUE if all of the elements in the tuple are TRUE, otherwise FALSE
sorted(t)	list	Sorts the tuple and returns a list

LIST METHODS

Operator	Returns	Comments
list()	list	Returns a new empty list. You can also use [] to initialize a new empty list
list(sequence)	list	Returns new list initialized from sequence’s items
[item [,item]+]	list	Writing a number of comma-separated items in square brackets constructs a new list of those items
x+y	list	Returns a new list containing the concatenation of the items in x and y
e in x	bool	Returns True if the item e is in x and False otherwise
del x[i]	bool	Deletes the item at index i in x. This is not an expression and does not return a value
x==y	bool	Returns True if x and y contain the same number of items and each of those corresponding items are pairwise equal
x>=y	bool	Returns True if x is greater than or equal to y according to a lexicographical ordering of the elements in x and y. If x and y have different lengths their items are == up to the shortest length, then this returns True if x is longer than y
x<=y	bool	Returns True if x is lexicographically before y or equal to y and False otherwise
x>y	bool	Returns True if x is lexicographically after y and False otherwise
x<y	bool	Returns True if x is lexicographically before y and False otherwise
x!=y	bool	Returns True if x and y are of different length or if some item of x is not == to some item of y. Otherwise it returns False
x[i]	item	Returns the item at index i of x
x[[i]:[j]]	list	Returns the slice of items starting at index i and extending to index j−1 in the string. If i is omitted then the slice begins at index 0. If j is omitted then the slice extends to the end of the list. If i is negative then it returns the slice starting at index len(x)+i (and likewise for the slice ending at j)
x[i]=e		Assigns the position at index i the value of e in x. The list x must already have an item at index i before this assignment occurs. In other words, assigning an item to a list in this way will not extend the length of the list to accommodate it
x+=y		This mutates the list x to append the items in y
x*=i		This mutates the list x to be i copies of the original x
iter(x)	list	Returns an iterator over x
len(x)	list	Returns the number of items in x
x*i	list	Returns a new list with the items of x repeated i times
i*x	list	Returns a new list with the items of x repeated i times
repr(x)	str	Returns a string representation of x
x.append(e)	none	This mutates the value of x to add e as its last element. The function returns None, but the return value is irrelevant since it mutates x
x.count(e)	int	Returns the number of occurrences of e in x by using == equality
x.extend(iter)	none	Mutates x by appending elements from the iterable, iter
x.index(e,[i,[j]])	int	Returns the first index of an element that == e between the start index, i, and the stop index, j−1. It raises ValueError if the value is not present in the specified sequence. If j is omitted then it searches to the end of the list. If i is omitted then it searches from the beginning of the list
x.insert(i, e)	none	Insert e before index i in x, mutating x
x.pop([index])	item	Remove and return the item at index. If index is omitted then the item at len(x)−1 is removed. The pop method returns the item and mutates x. It raises IndexError if list is empty or index is out of range
x.remove(e)	none	remove first occurrence of e in x, mutating x. It raises ValueError if the value is not present
x.reverse()	none	Reverses all the items in x, mutating x
x.sort()	none	Sorts all the items of x according to their natural ordering as determined by the item’s _ _cmp_ _ method, mutating x. Two keyword parameters are possible: key and reverse. If reverse = True is specified, then the result of sorting will have the list in reverse of the natural ordering. If key = f is specified then f must be a function that takes an item of x and returns the value of that item that should be used as the key when sorting

SET METHODS

Operator	Returns	Comments
s1 & s2	set	returns the intersection of two sets
s1 | s2	set	returns the union of two sets
s1.intersect(s1)	set	returns the intersection of two sets
s1.union(s2)	set	returns the union of two sets
s1.difference(s2)	set	returns the elements in set s1 that do not exist in set s2
s1.issubset(s2)	bool	returns TRUE if all the elements of the set s1 exist in set s2
s1.issuperset(s2)	bool	returns TRUE if all the elements of the set s2 exist in set s1
s.update(t)	set	return set s with elements added from t
s.intersection_update(t)	set	return set s keeping only elements also found in t
s.difference_update(t)	set	return set s after removing elements found in t
s.symmetric_difference_update(t)	set	return set s with elements from s or t but not both
s.add(x)	set	add element x to set s
s.remove(x)	set	remove x from set s; raises KeyError if not present
s.discard(x)	set	removes x from set s if present
s.pop()		remove and return an arbitrary element from s; raises KeyError if empty
s.clear()		remove all elements from set s

DICTIONARY METHODS

Operator	Returns	Comments
dict()	dict	New empty dictionary
dict(mapping)	dict	New dictionary initialized from a mapping object’s (key, value) pairs
dict(seq)	dict	New dictionary initialized as if via D = {} for k, v in seq D[k] = v
dict(**kwargs)	dict	New dictionary initialized with the name = value pairs in the keyword arg list. For example: dict(one = 1, two = 2)
k in D	bool	True if D has key k, else False
del D[k]		Deletes key k from dictionary D
D1== 2	bool	Returns True if dictionaries D1 and D2 have same keys mapped to same values
D[k]	value type	Returns value k maps to in D. If k is not mapped, it raises a KeyError exception
iter(D)	iterator	Returns an iterator over D
len(D)	int	Returns the number of keys in D
D1!=D2	bool	Returns True if D1 and D2 have any different keys or keys map to different values
repr(D)	str	Returns a string representation of D
D[k]=e	-	Stores the key,value pair k,e in D

Method	Returns	Comments
D.clear()	none	Remove all items from D
D.copy()	dict	A shallow copy of D
D.get(k[,e])	value type	D[k] if k in D, else e. e defaults to None
D.items()	items	A set-like object providing a view on D’s items
D.keys()	keys	A set-like object providing a view on D’s keys
D.pop(k[,e])	v	Remove specified key and return the corresponding value. If key is not found, e is returned if given, otherwise KeyError is raised
D.popitem()	(k,v)	Remove and return some (key, value) pair as a 2-tuple; but raise KeyError if D is empty
D.setdefault(k[,e])	D.get(k,e)	Returns D.get(k,e) and also sets d[k] = e if k not in D
D.update(E, **F)	none	Update D from dict/iterable E and F If E has a .keys() method, does: for k in E: D[k] = E[k] If E lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
D.values()	values	An object providing a view on D’s values

IF STATEMENTS

- A colon is required after the logical expression
- There is no explicit block, the block is implicitly defined by indentation
- The block might have one or more lines

Examples

Simplest form

```
if <logical expression>:
  statement(s)
```

```
x1 = 3
x2 = 5
if x1<x2: print('greater')
```

The form with “else”

```
if <logical expression>:
  statement(s)
else:
  statement(s)
```

```
x1 = 3
x2 = 5
if x1>x2: print('greater')
else: print('not greater')
```

Nested if statements, i.e. “elif”

```
if <logical expression>:
 statement(s)
elif <logical expression>:
 statement(s)
else <logical expression>:
 statement(s)
```

```
x1 = 3
x2 = 3
if x1>x2: print('greater')
elif x1==x2: print('equal')
else: print('not greater')
```

FUNCTIONS

Examples

range()

```
range(n)
range(n_start,n_end)
range(n_start,n_end,step)
```

```
range(3)          # 0,1,2
range(10,13)      # 10,11,12
range(10,20,3)    # 10,13,16,19
```

enumerate()

```
enumerate(n)
```

```
a = ['İstanbul', 'Ankara', 'İzmir']
list(enumerate(a))

# [(0, 'İstanbul'), (1, 'Ankara'), (2, 'İzmir')]
```

WHILE LOOP

while loop

- Looping while a logical condition holds true.
- The condition is checked before looping.
- Multiple lines are allowed.

```
while <logical expression>:
  statement(s)
```

Example:

```
result = 0
i=0
while i<100:
    result += i
    i += 1
result
```

- **break** statement terminates the loop
- **continue** statement does not terminate the loop but terminates the current iteration

```
result = 0
i=0
while i<100:
    if result>250:
        break
    result += i
    i += 1
result
```

FOR LOOP

for loop

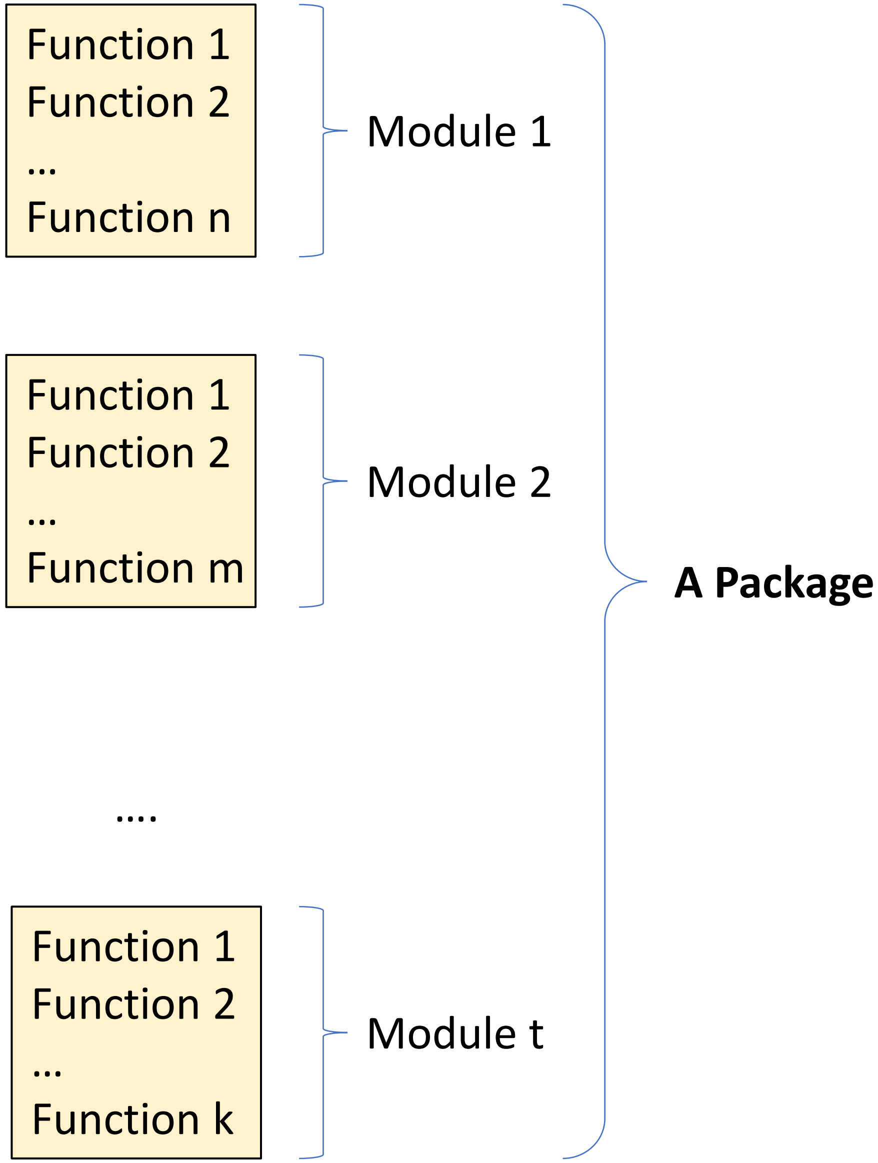
- Looping while a set of values are traced.
- The set is checked before looping.
- Multiple lines are allowed.

```
for i in <sequence or a set of values>:
  statement(s)
```

<u>Example:</u>

```
result = 0
for i in range(100):
    result += i
result
```

FUNCTIONS and PACKAGES



Function 1
Function 2
…
Function n

Module 1

Function 1
Function 2
…
Function m

Module 2

….

Function 1
Function 2
…
Function k

Module t

A Package

Function examples

```
def MySum (a,b):
    return a+b
```

```
def MySumProd (a,b):
    return a+b, a*b
```

Installing packages

Option 1: Use conda. It is an environment manager.
It could install other applications too.

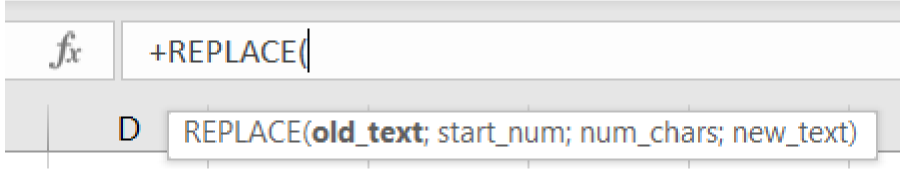
```
conda install package_name
```

Option 2: Use pip. It is default package installation tool.

```
pip install package_name
```

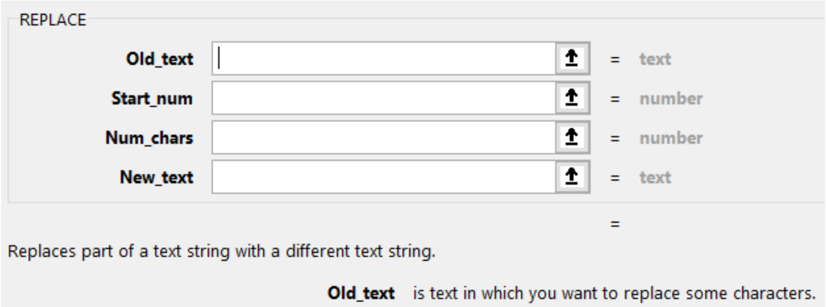
OBJECT ORIENTED LANGUAGE

	Data	**Functions**
Classical languages	MyStr = ‘customer'	Replace(String, FindStr, ReplaceStr)



+REPLACE(

REPLACE(**old_text**; start_num; num_chars; new_text)



REPLACE

Old_text = text
Start_num = number
Num_chars = number
New_text = text

=

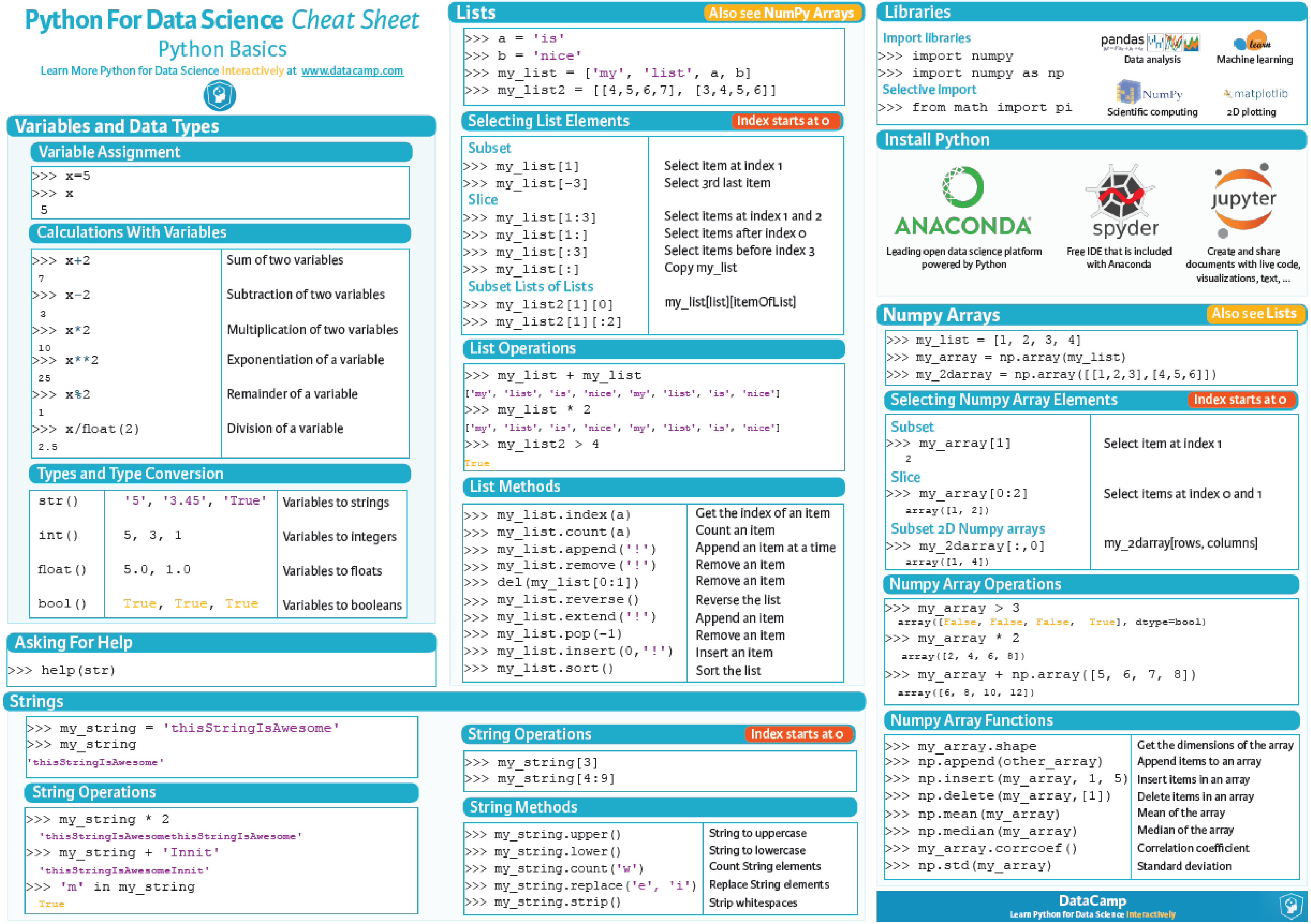
Replaces part of a text string with a different text string.

Old_text is text in which you want to replace some characters.

- *Data and functions are independent of each other*
- *The variable stores only data*

Object Oriented languages	MyStr = ‘customer'	MyStr.replace('c', 'C')

- *Data and functions are being part of an object*
- *Each object belongs to a **class** (i.e., type). An object is a realization or instantiation of a class.*
- *First the class is defined together with its **method**s (i.e. functions)*
- *Then an object is created as an “instance” of that class*
- *More objects of the same class could be created*



Python For Data Science *Cheat Sheet*

Python Basics

Learn More Python for Data Science Interactively at www.datacamp.com

Variables and Data Types

Variable Assignment

```
>>> x=5
>>> x
 5
```

Calculations With Variables

`>>> x+2`
`7`
	Sum of two variables

`>>> x-2`
`3`
	Subtraction of two variables

`>>> x*2`
`10`
	Multiplication of two variables

`>>> x**2`
`25`
	Exponentiation of a variable

`>>> x%2`
`1`
	Remainder of a variable

`>>> x/float(2)`
`2.5`
	Division of a variable

Types and Type Conversion

`str()`	`'5', '3.45', 'True'`	Variables to strings
`int()`	`5, 3, 1`	Variables to integers
`float()`	`5.0, 1.0`	Variables to floats
`bool()`	`True, True, True`	Variables to booleans

Asking For Help

```
>>> help(str)
```

Lists

Also see NumPy Arrays

```
>>> a = 'is'
>>> b = 'nice'
>>> my_list = ['my', 'list', a, b]
>>> my_list2 = [[4,5,6,7], [3,4,5,6]]
```

Selecting List Elements

Index starts at 0

Subset	
`>>> my_list[1]`	Select item at index 1
`>>> my_list[-3]`	Select 3rd last item
Slice	
`>>> my_list[1:3]`	Select items at index 1 and 2
`>>> my_list[1:]`	Select items after index 0
`>>> my_list[:3]`	Select items before index 3
`>>> my_list[:]`	Copy my_list
Subset Lists of Lists	

`>>> my_list2[1][0]`
`>>> my_list2[1][:2]`
	my_list[list][itemOfList]

List Operations

```
>>> my_list + my_list
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list * 2
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list2 > 4
True
```

List Methods

`>>> my_list.index(a)`	Get the index of an item
`>>> my_list.count(a)`	Count an item
`>>> my_list.append('!')`	Append an item at a time
`>>> my_list.remove('!')`	Remove an item
`>>> del(my_list[0:1])`	Remove an item
`>>> my_list.reverse()`	Reverse the list
`>>> my_list.extend('!')`	Append an item
`>>> my_list.pop(-1)`	Remove an item
`>>> my_list.insert(0,'!')`	Insert an item
`>>> my_list.sort()`	Sort the list

Strings

```
>>> my_string = 'thisStringIsAwesome'
>>> my_string
'thisStringIsAwesome'
```

String Operations

```
>>> my_string * 2
 'thisStringIsAwesomethisStringIsAwesome'
>>> my_string + 'Innit'
 'thisStringIsAwesomeInnit'
>>> 'm' in my_string
 True
```

String Operations

Index starts at 0

```
>>> my_string[3]
>>> my_string[4:9]
```

String Methods

`>>> my_string.upper()`	String to uppercase
`>>> my_string.lower()`	String to lowercase
`>>> my_string.count('w')`	Count String elements
`>>> my_string.replace('e', 'i')`	Replace String elements
`>>> my_string.strip()`	Strip whitespaces

Libraries

Import libraries

```
>>> import numpy
>>> import numpy as np
```

Selective import

```
>>> from math import pi
```

pandas — Data analysis

learn — Machine learning

NumPy — Scientific computing

matplotlib — 2D plotting

Install Python

ANACONDA — Leading open data science platform powered by Python

spyder — Free IDE that is included with Anaconda

jupyter — Create and share documents with live code, visualizations, text, ...

Numpy Arrays

Also see Lists

```
>>> my_list = [1, 2, 3, 4]
>>> my_array = np.array(my_list)
>>> my_2darray = np.array([[1,2,3],[4,5,6]])
```

Selecting Numpy Array Elements

Index starts at 0

Subset	

`>>> my_array[1]`
`2`
	Select item at index 1

Slice	

`>>> my_array[0:2]`
`array([1, 2])`
	Select items at index 0 and 1

Subset 2D Numpy arrays	

`>>> my_2darray[:,0]`
`array([1, 4])`
	my_2darray[rows, columns]

Numpy Array Operations

```
>>> my_array > 3
  array([False, False, False,  True], dtype=bool)
>>> my_array * 2
  array([2, 4, 6, 8])
>>> my_array + np.array([5, 6, 7, 8])
  array([6, 8, 10, 12])
```

Numpy Array Functions

`>>> my_array.shape`	Get the dimensions of the array
`>>> np.append(other_array)`	Append items to an array
`>>> np.insert(my_array, 1, 5)`	Insert items in an array
`>>> np.delete(my_array,[1])`	Delete items in an array
`>>> np.mean(my_array)`	Mean of the array
`>>> np.median(my_array)`	Median of the array
`>>> my_array.corrcoef()`	Correlation coefficient
`>>> np.std(my_array)`	Standard deviation

DataCamp

Learn Python for Data Science Interactively

AA applied analytics

WHAT WE HAVE LEARNED AT THIS LECTURE?

INSPIRATION

REASONING

INTUITION

PRACTICE

PRINCIPLE

CONCLUSION

MATH

THINKING

CODING SKILL

USE CASE

FORMULA

IDEA

THEORY