BMI 503 Computer Science for Biomedical Informatics

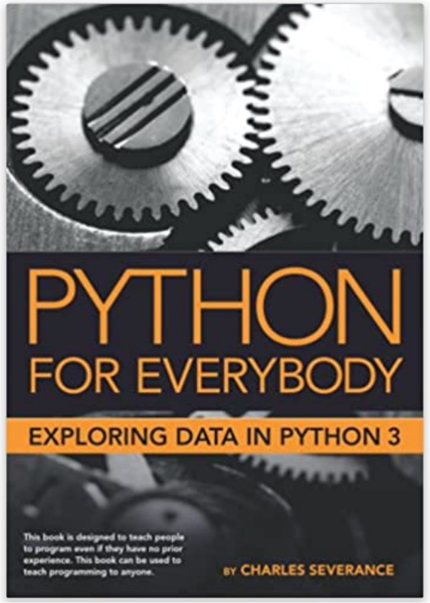
Chao Chen

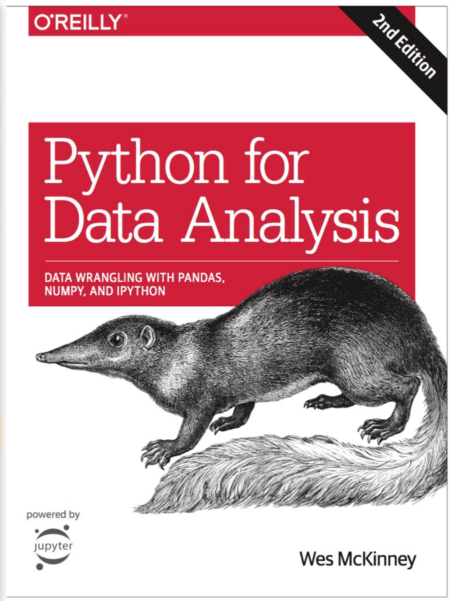
Stony Brook University

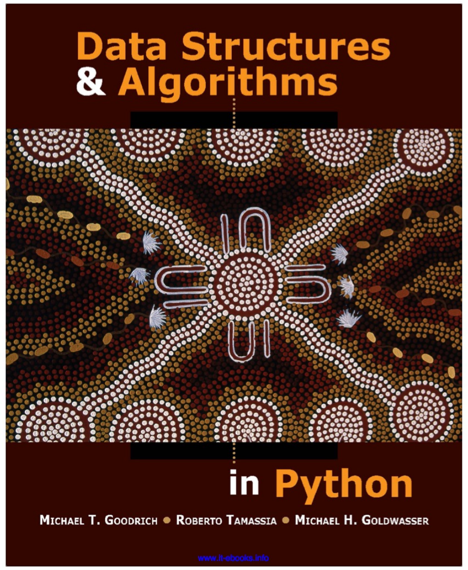
Aug, 2022

Overview

- **Audience:** students with limited/no background of computation.
- **Goal:** To prepare students for basic programming tasks and data analytics courses.
- Instructor: Chao Chen, chao.chen.1@stonybrook.edu
- Time: Mon 10:00am to 12:30pm, in-person
- Office Hour: Mon 1-2, Th/Fr?
- Key content: Basic python programming; Data structures
- **Reference books:**
  - Python for Everybody: Exploring Data in Python 3 https://www.py4e.com/
  - Python for Data Analysis: Data Wrangling with Pandas, NumPy, and Ipython Book by Wes McKinney
  - Data Structures and Algorithms in Python: Michael T. Goodrich et al.






Data Structures & Algorithms
in Python
Michael T. Goodrich • Roberto Tamassia • Michael H. Goldwasser

Content

-------- Python programming essence -----------

* variables, loops, functions, object-oriented programming.

* running the code in a development environment, debug the code.

* IO, reading/writing files, string operations.

--------- Data structures and basic programming projects -------------

* list, array, stack, queue, tree, heap

* sorting, dynamic programming

* basic complexity analysis

Evaluation

- Mid-term: 30 pts (in class, TBD).
- Final: 30 pts (TBD).
- Programming projects (2 - 3, workload varies): 40 pts.

Round Table: Self-Introduction

- Name
- Department/Program
- Why taking this course?
- Background in Programming?

- If you feel comfortable with programming, you can skip classes and just let me know.

Lecture 1. Python Primer – part 1

Chao Chen

Stony Brook University

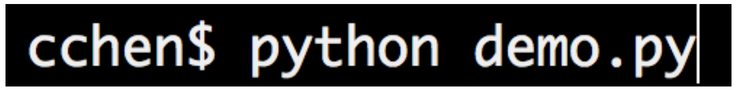
Aug 22, 2022

Python

Program – a piece of code performing certain functionality

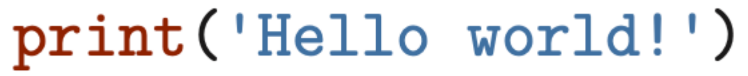
3 ways to run a python program

- Interactive environment
  - Python/iPython – directly typing the command
    quit() to exit
  - jupyter notebook – interactive environment with fancier browser interface (figures, tables, etc)
  - Good for prototyping, demo; Does not scale
- Executing one or multiple program files (*.py)

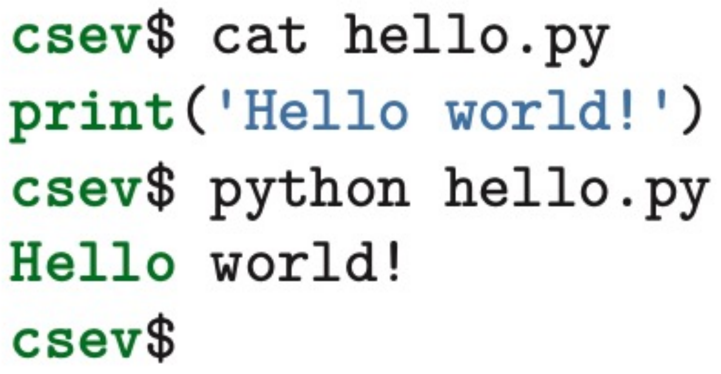


```
cchen$ python demo.py
```

  - Python/iPython: exec(open("filename.py").read())
  - Jupyter notebook
    %load filename.py
- Integrated Development Environments (IDE)
  - Debugging
  - Scalable: managing multiple files / functions
  - Less intuitive



```
print('Hello world!')
```



```
csev$ cat hello.py
print('Hello world!')
csev$ python hello.py
Hello world!
csev$
```

Grammar of python programs

```
print('Hello world!')
```

```
>>> print('You must be the legendary god that comes from the sky')
You must be the legendary god that comes from the sky
>>> print('We have been waiting for you for a long time')
We have been waiting for you for a long time
>>> print('Our legend says you will be very tasty with mustard')
Our legend says you will be very tasty with mustard
>>> print 'We will have a feast tonight unless you say
  File "<stdin>", line 1
    print 'We will have a feast tonight unless you say
                                                     ^

SyntaxError: Missing parentheses in call to 'print'
>>>
```

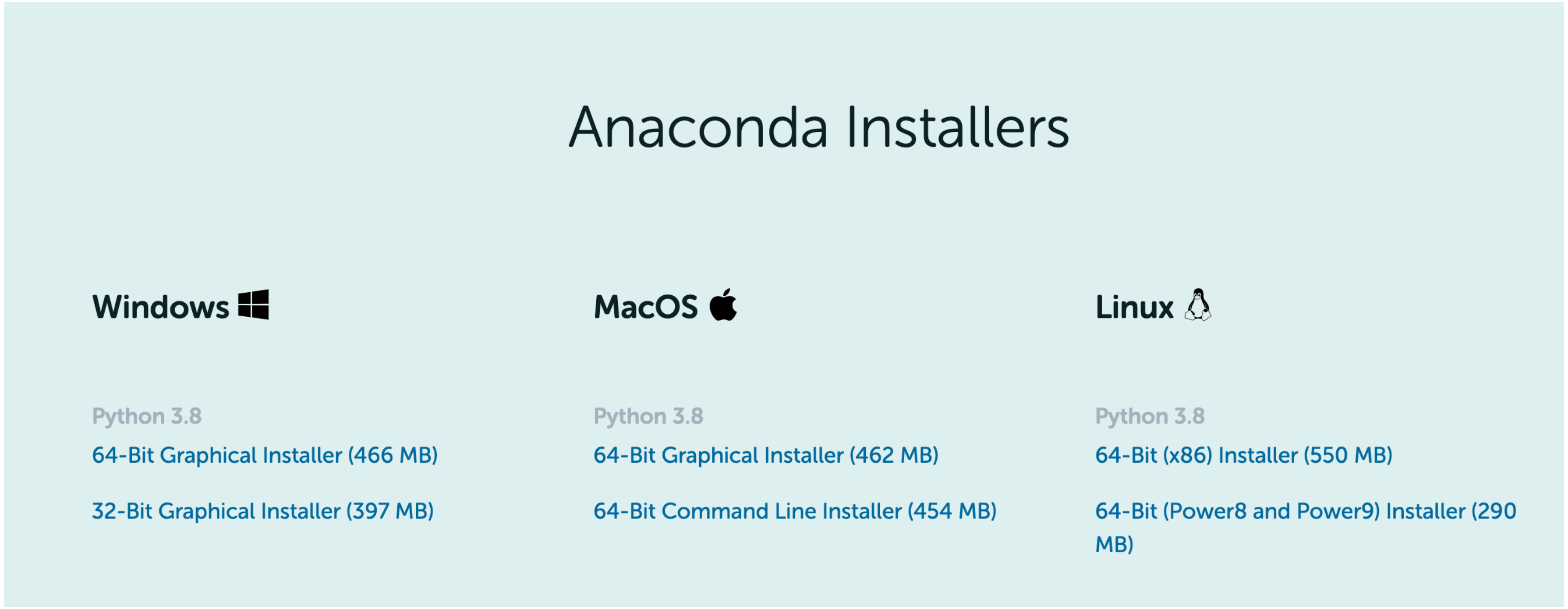
Python

```
print('Welcome to the GPA calculator.')
print('Please enter all your letter grades, one per line.')
print('Enter a blank line to designate the end.')
# map from letter grade to point value
points = {'A+':4.0, 'A':4.0, 'A-':3.67, 'B+':3.33, 'B':3.0, 'B-':2.67,
          'C+':2.33, 'C':2.0, 'C':1.67, 'D+':1.33, 'D':1.0, 'F':0.0}
num_courses = 0
total_points = 0
done = False
while not done:
  grade = input( )                                      # read line from user
  if grade == '':                                       # empty line was entered
    done = True
  elif grade not in points:                             # unrecognized grade entered
    print("Unknown grade '{0}' being ignored".format(grade))
  else:
    num_courses += 1
    total_points += points[grade]
if num_courses > 0:                                     # avoid division by zero
  print('Your GPA is {0:.3}'.format(total_points / num_courses))
```

Code Fragment 1.1: A Python program that computes a grade-point average (GPA).

Installation: Anaconda

- Anaconda – a comprehensive python distribution and easy to maintain https://www.anaconda.com/products/individual

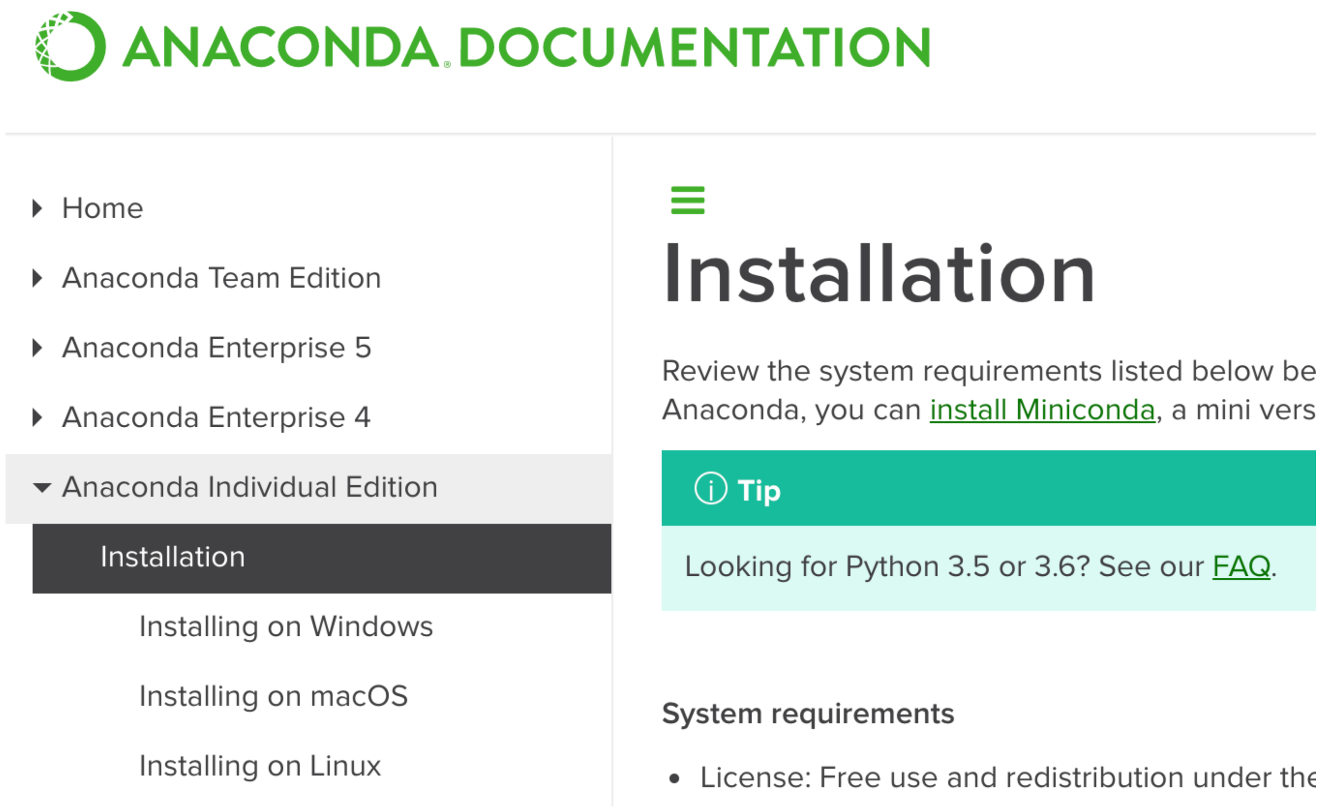


Anaconda Installers

Windows	MacOS	Linux
Python 3.8	Python 3.8	Python 3.8
64-Bit Graphical Installer (466 MB)	64-Bit Graphical Installer (462 MB)	64-Bit (x86) Installer (550 MB)
32-Bit Graphical Installer (397 MB)	64-Bit Command Line Installer (454 MB)	64-Bit (Power8 and Power9) Installer (290 MB)

Installation: Anaconda (cont'd)

- Installation guide: https://docs.anaconda.com/anaconda/install/


ANACONDA DOCUMENTATION

- Home
- Anaconda Team Edition
- Anaconda Enterprise 5
- Anaconda Enterprise 4
- Anaconda Individual Edition
  - Installation
    - Installing on Windows
    - Installing on macOS
    - Installing on Linux

Installation

Review the system requirements listed below be
Anaconda, you can install Miniconda, a mini vers

Tip

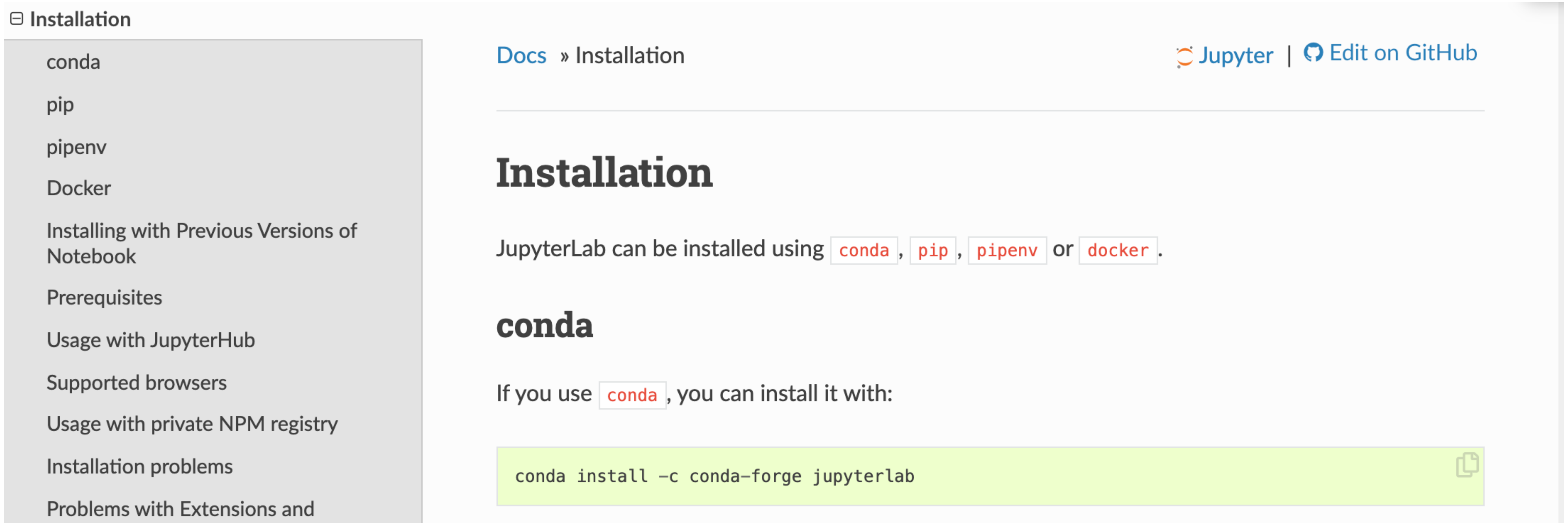
Looking for Python 3.5 or 3.6? See our FAQ.

System requirements

- License: Free use and redistribution under the

Installation: Jupyter Notebook

- Jupyter notebook (installed via anaconda) https://jupyterlab.readthedocs.io/en/stable/getting_started/installation.html



⊟ Installation

- conda
- pip
- pipenv
- Docker
- Installing with Previous Versions of Notebook
- Prerequisites
- Usage with JupyterHub
- Supported browsers
- Usage with private NPM registry
- Installation problems
- Problems with Extensions and

Docs » Installation

Jupyter | Edit on GitHub

Installation

JupyterLab can be installed using `conda`, `pip`, `pipenv` or `docker`.

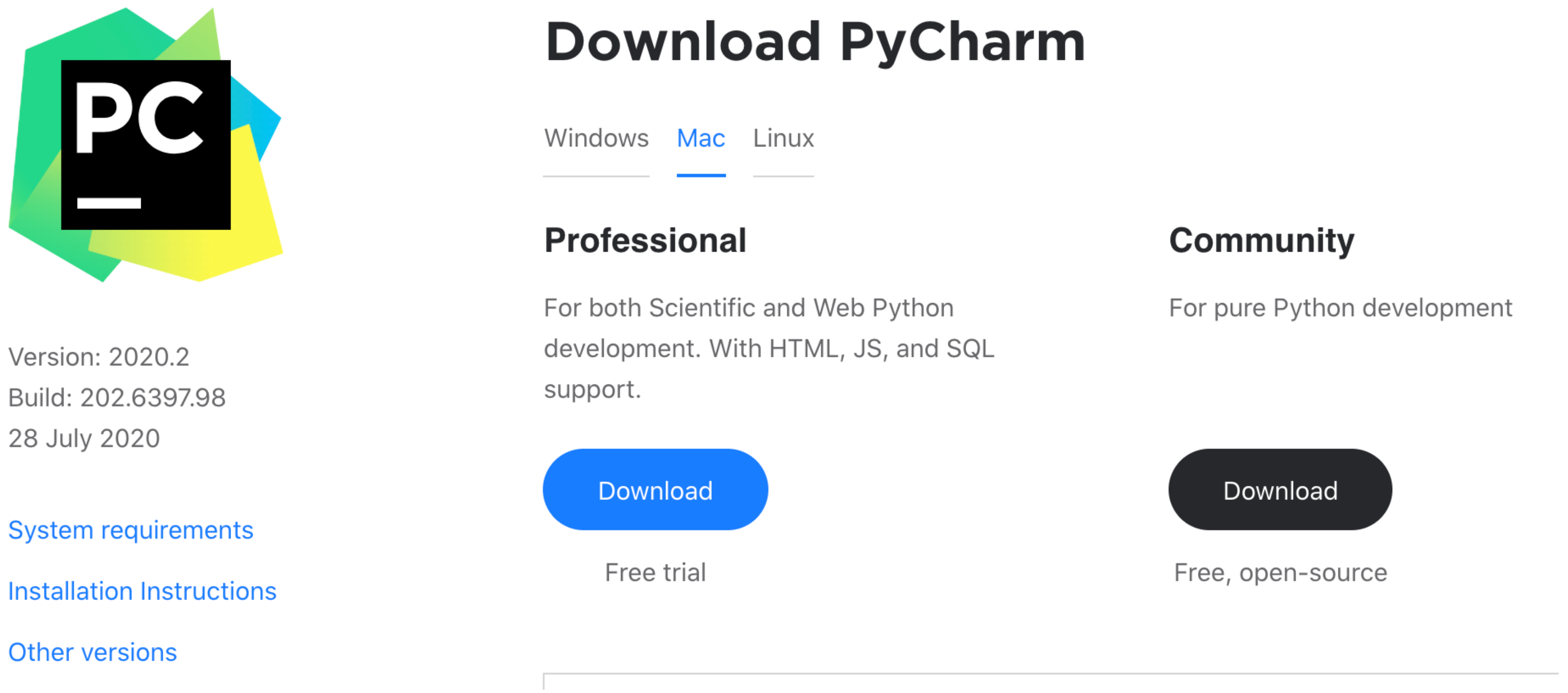
conda

If you use `conda`, you can install it with:

```
conda install -c conda-forge jupyterlab
```

Installation: IDE

- PyCharm (and many others, whatever you like) https://www.jetbrains.com/pycharm/



Download PyCharm

Windows Mac Linux

Professional

For both Scientific and Web Python development. With HTML, JS, and SQL support.

Download

Free trial

Community

For pure Python development

Download

Free, open-source

Version: 2020.2
Build: 202.6397.98
28 July 2020

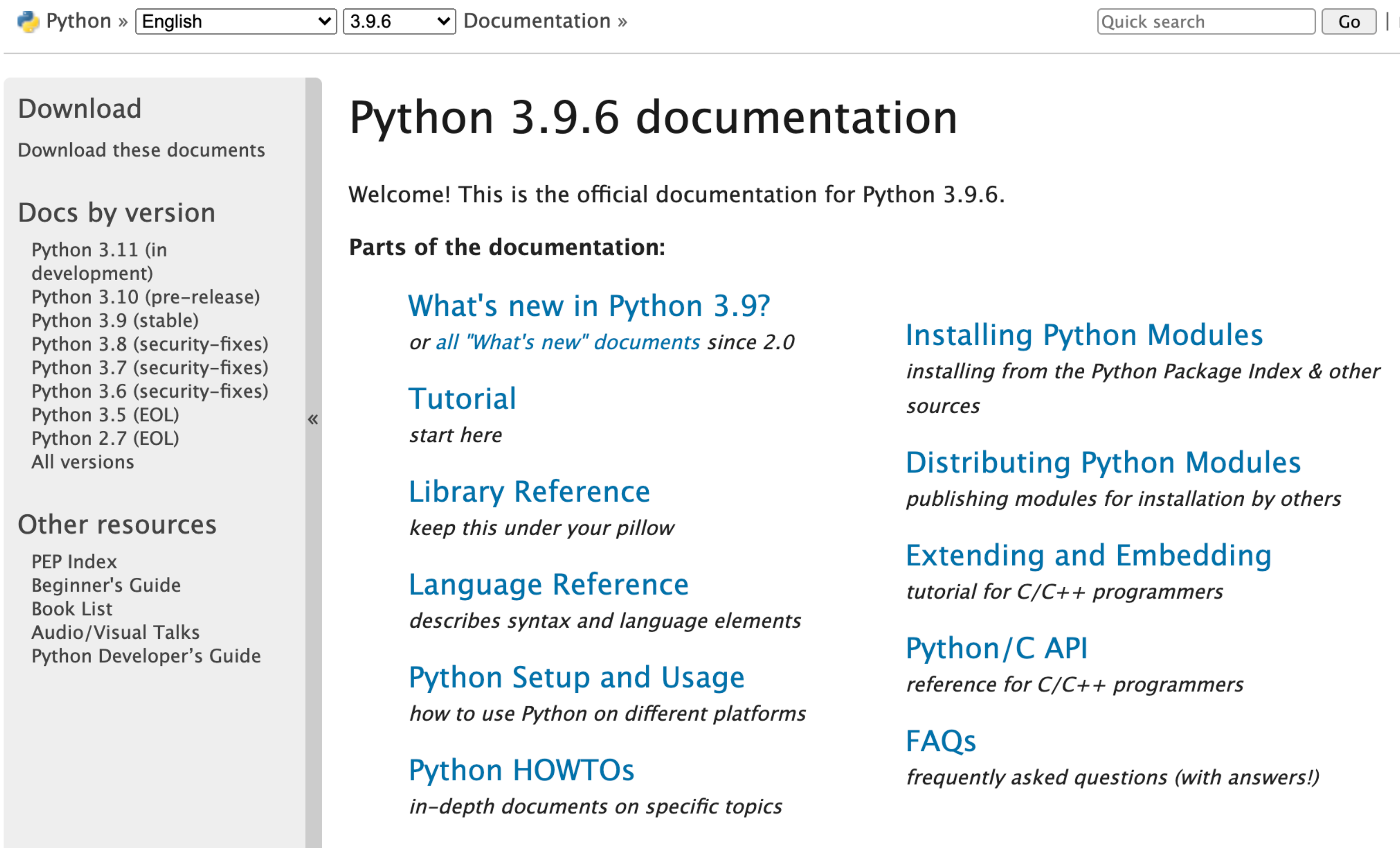
System requirements

Installation Instructions

Other versions

Python

- Disclaimer: take the lecture with a grain of salt.
- Python is a very dynamic language
  - Open source, changes are made all the time
  - Proceed with caution
  - Always try first
  - Learn to read the reference https://docs.python.org/3/



Download

Download these documents

Docs by version

Python 3.11 (in development)
Python 3.10 (pre-release)
Python 3.9 (stable)
Python 3.8 (security-fixes)
Python 3.7 (security-fixes)
Python 3.6 (security-fixes)
Python 3.5 (EOL)
Python 2.7 (EOL)
All versions

Other resources

PEP Index
Beginner's Guide
Book List
Audio/Visual Talks
Python Developer's Guide

Python 3.9.6 documentation

Welcome! This is the official documentation for Python 3.9.6.

Parts of the documentation:

What's new in Python 3.9?
or all "What's new" documents since 2.0

Tutorial
start here

Library Reference
keep this under your pillow

Language Reference
describes syntax and language elements

Python Setup and Usage
how to use Python on different platforms

Python HOWTOs
in-depth documents on specific topics

Installing Python Modules
installing from the Python Package Index & other sources

Distributing Python Modules
publishing modules for installation by others

Extending and Embedding
tutorial for C/C++ programmers

Python/C API
reference for C/C++ programmers

FAQs
frequently asked questions (with answers!)

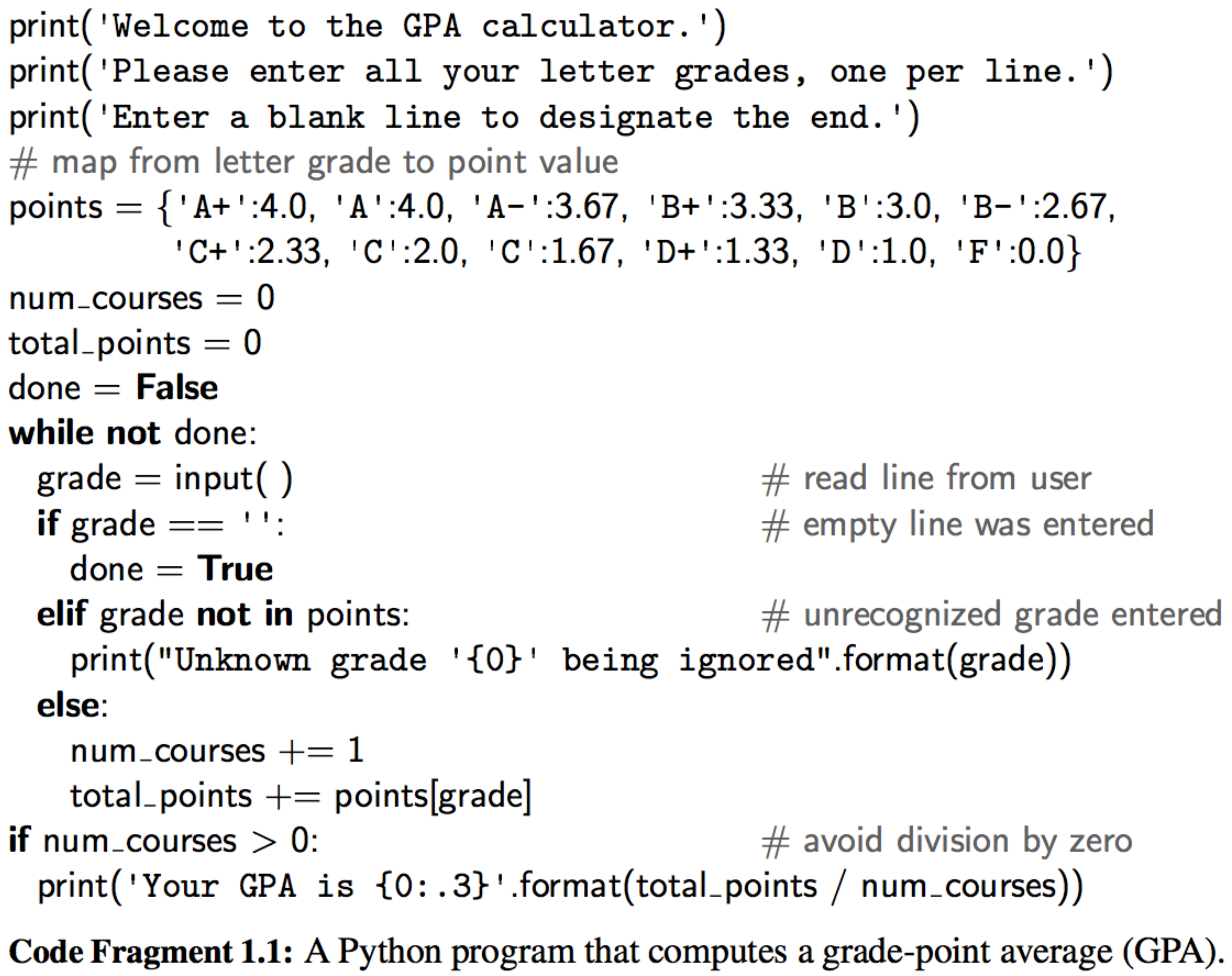
Lines & Commands

- Commands: single instructions to execute

One line

- Usually one command, finish with a line break
- If one command cannot fit one line, use ‘ \ ’ to extend
- Unfinished ‘ { [(‘ also works but I recommend not
- Semi-colon: separate commands in a same line
- White space (indentation) + colon: defines control sequences
- Comments: anything after ’ # ’ within the same line
- Reserved words (boldfaced)

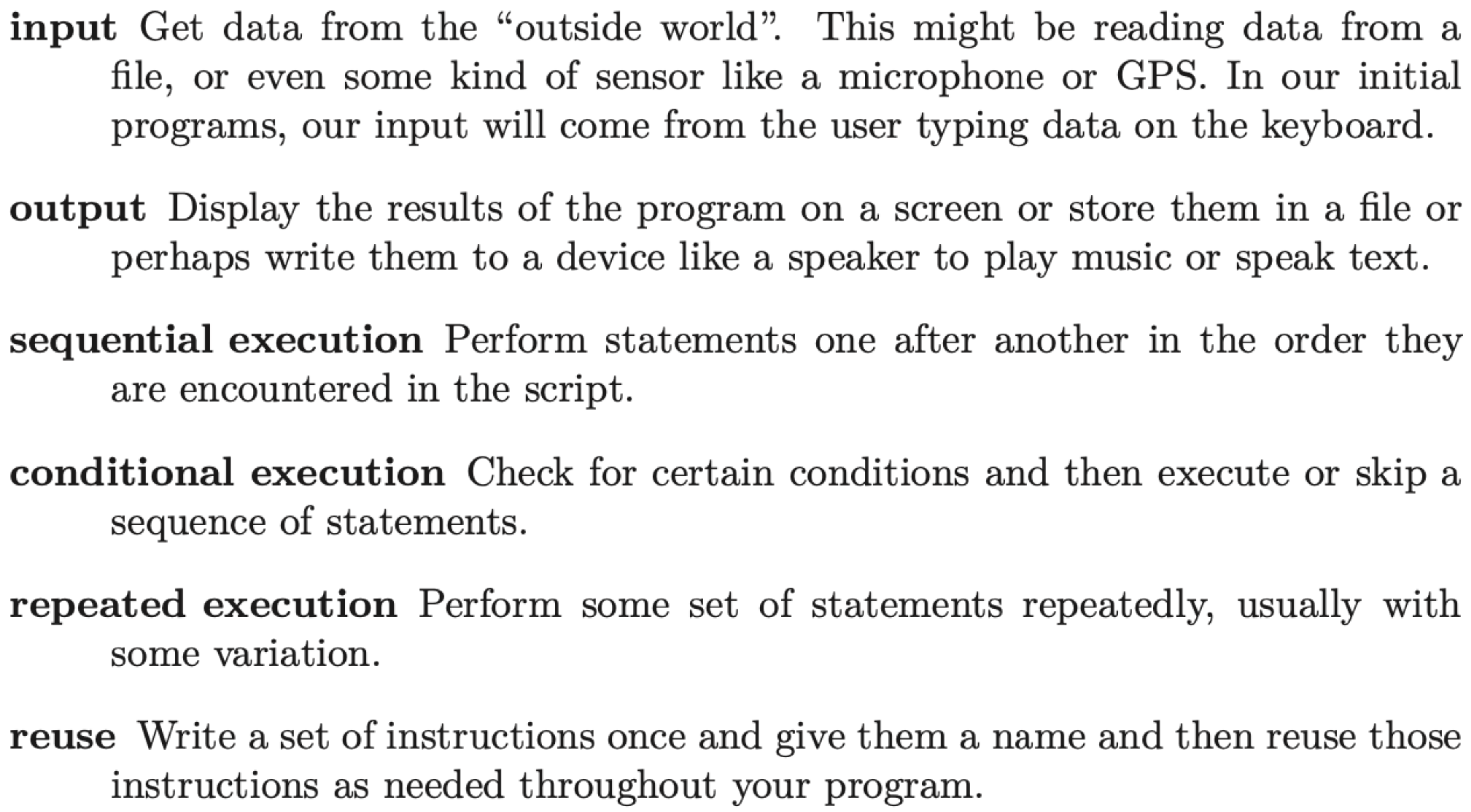
Similar principles apply to most programming languages: C++, Java, Matlab, etc.



```
print('Welcome to the GPA calculator.')
print('Please enter all your letter grades, one per line.')
print('Enter a blank line to designate the end.')
# map from letter grade to point value
points = {'A+':4.0, 'A':4.0, 'A-':3.67, 'B+':3.33, 'B':3.0, 'B-':2.67,
          'C+':2.33, 'C':2.0, 'C':1.67, 'D+':1.33, 'D':1.0, 'F':0.0}
num_courses = 0
total_points = 0
done = False
while not done:
  grade = input( )                          # read line from user
  if grade == '':                           # empty line was entered
    done = True
  elif grade not in points:                 # unrecognized grade entered
    print("Unknown grade '{0}' being ignored".format(grade))
  else:
    num_courses += 1
    total_points += points[grade]
if num_courses > 0:                         # avoid division by zero
  print('Your GPA is {0:.3}'.format(total_points / num_courses))
```

Code Fragment 1.1: A Python program that computes a grade-point average (GPA).

The building blocks of a program



input Get data from the "outside world". This might be reading data from a file, or even some kind of sensor like a microphone or GPS. In our initial programs, our input will come from the user typing data on the keyboard.

output Display the results of the program on a screen or store them in a file or perhaps write them to a device like a speaker to play music or speak text.

sequential execution Perform statements one after another in the order they are encountered in the script.

conditional execution Check for certain conditions and then execute or skip a sequence of statements.

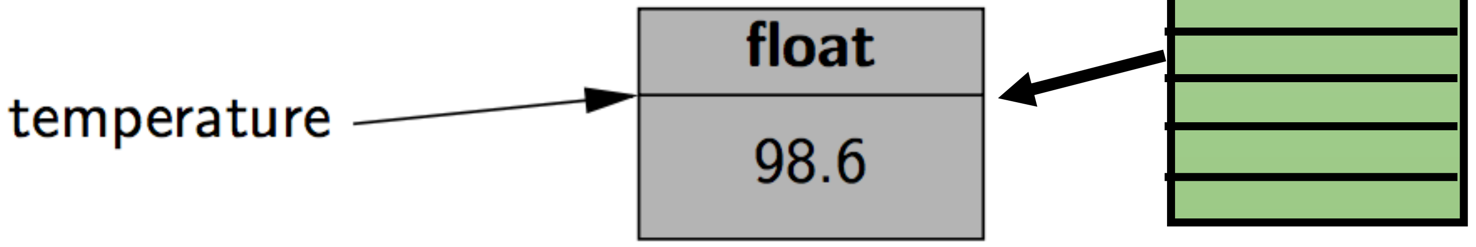
repeated execution Perform some set of statements repeatedly, usually with some variation.

reuse Write a set of instructions once and give them a name and then reuse those instructions as needed throughout your program.

Identifiers, Objects and the Assignment Statement

- Built-in classes (types): int, float, str, etc.
- Object: of a given class (type), with a specific value, stored in a particular memory address
- Identifier / name: pointing toward an object / memory address
- Assignment statement
  Initiate an object (also type/class), associate it with an identifier
- Note the type is not explicitly defined in the statement
  -- Python is dynamically-typed (Java, C++, etc, are not)
  -- Makes your life easier, but also more dangerous.

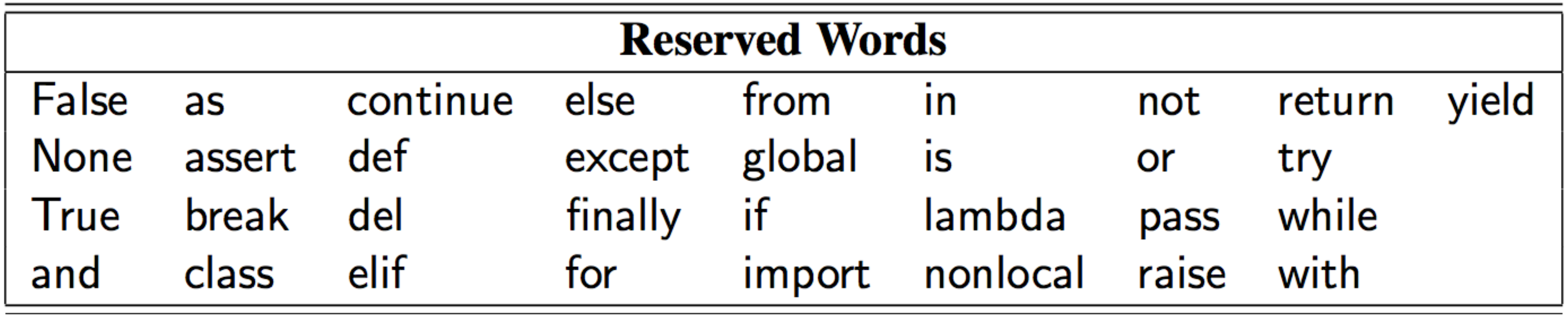
$$\text{temperature} = 98.6$$



More about Identifiers

- Case-sensitive: temperature and Temperature are different
- Combinations of letters, numbers, underscore

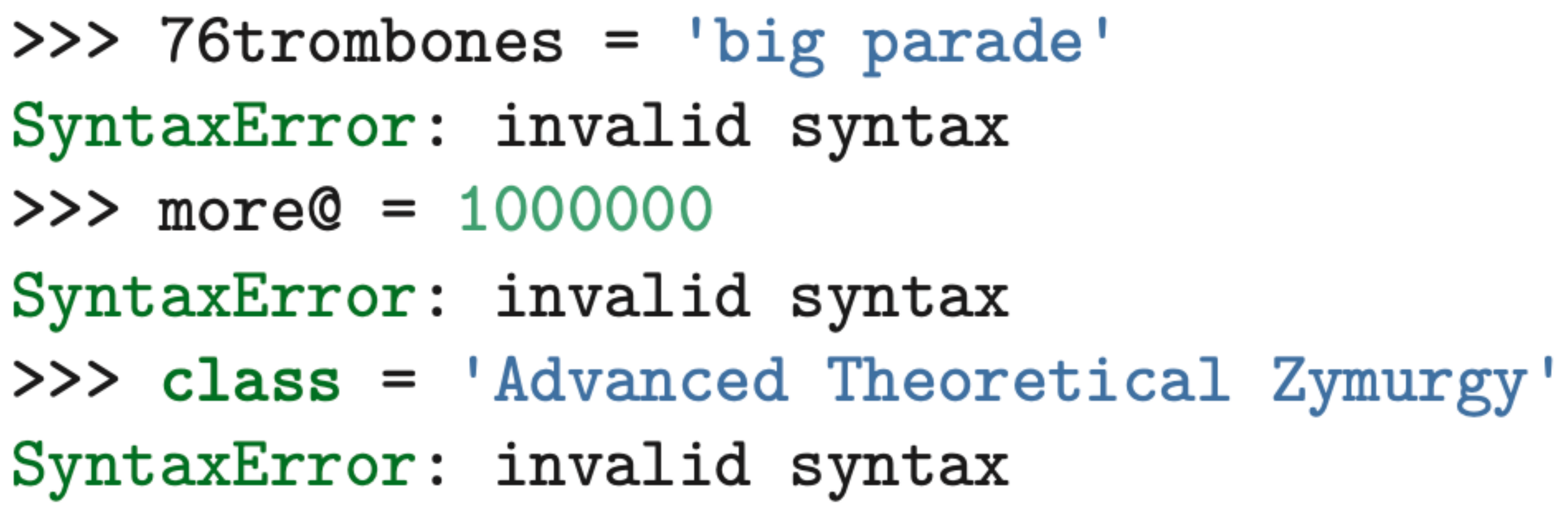
  `my_name` or `airspeed_of_unladen_swallow`.
- Can not start with numbers
- Can not use reserved words



Reserved Words								
False	as	continue	else	from	in	not	return	yield
None	assert	def	except	global	is	or	try	
True	break	del	finally	if	lambda	pass	while	
and	class	elif	for	import	nonlocal	raise	with	

More about Identifiers

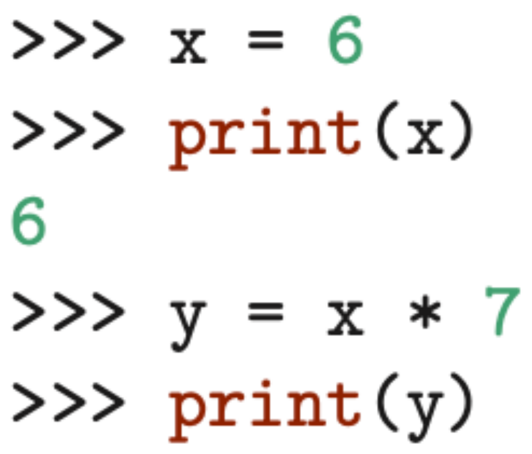
- What are the reasons for the following errors?



```
>>> 76trombones = 'big parade'
SyntaxError: invalid syntax
>>> more@ = 1000000
SyntaxError: invalid syntax
>>> class = 'Advanced Theoretical Zymurgy'
SyntaxError: invalid syntax
```

Another example

- Another example



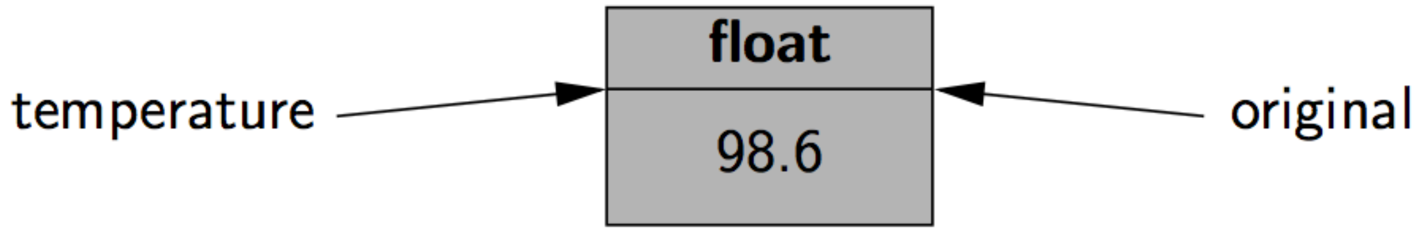
```
>>> x = 6
>>> print(x)
6
>>> y = x * 7
>>> print(y)
```

What is the output?

Alias

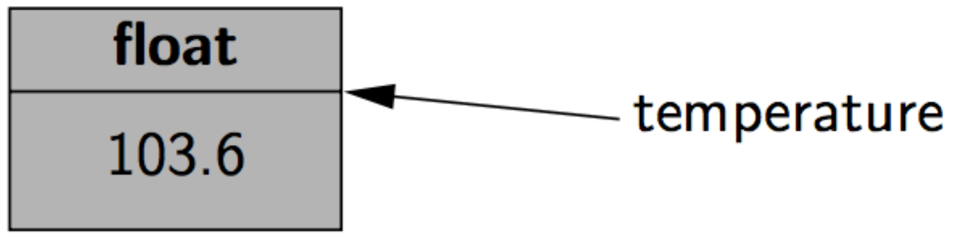
- Multiple identifiers pointing toward a same object

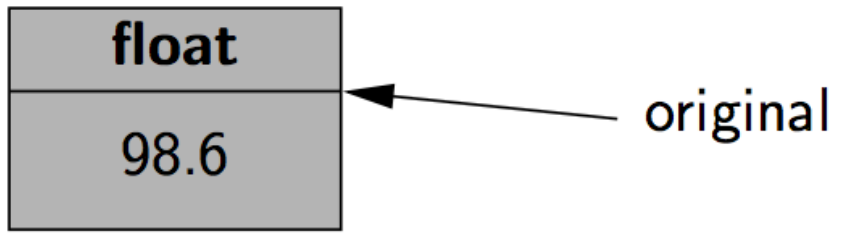
```
original = temperature.
```



- Reassignment: `temperature = temperature + 5.0`

  - evaluate expression on right hand side
  - create a new object, and associate “temperature” to it.
  - old object unaffected (with the alias still associated)





More examples

Exercise 5: What is wrong with the following code:

```
>>> primt 'Hello world!'
  File "<stdin>", line 1
    primt 'Hello world!'
                       ^
SyntaxError: invalid syntax
>>>
```

Exercise 7: What will the following program print out:

```
x = 43
x = x + 1
print(x)
```

Values and Types

```
>>> type('Hello, World!')
<class 'str'>
>>> type(17)
<class 'int'>
>>> type(3.2)
<class 'float'>
```

```
>>> type('17')

>>> type('3.2')

>>> print(1,000,000)

```

Variables, assignments and Types

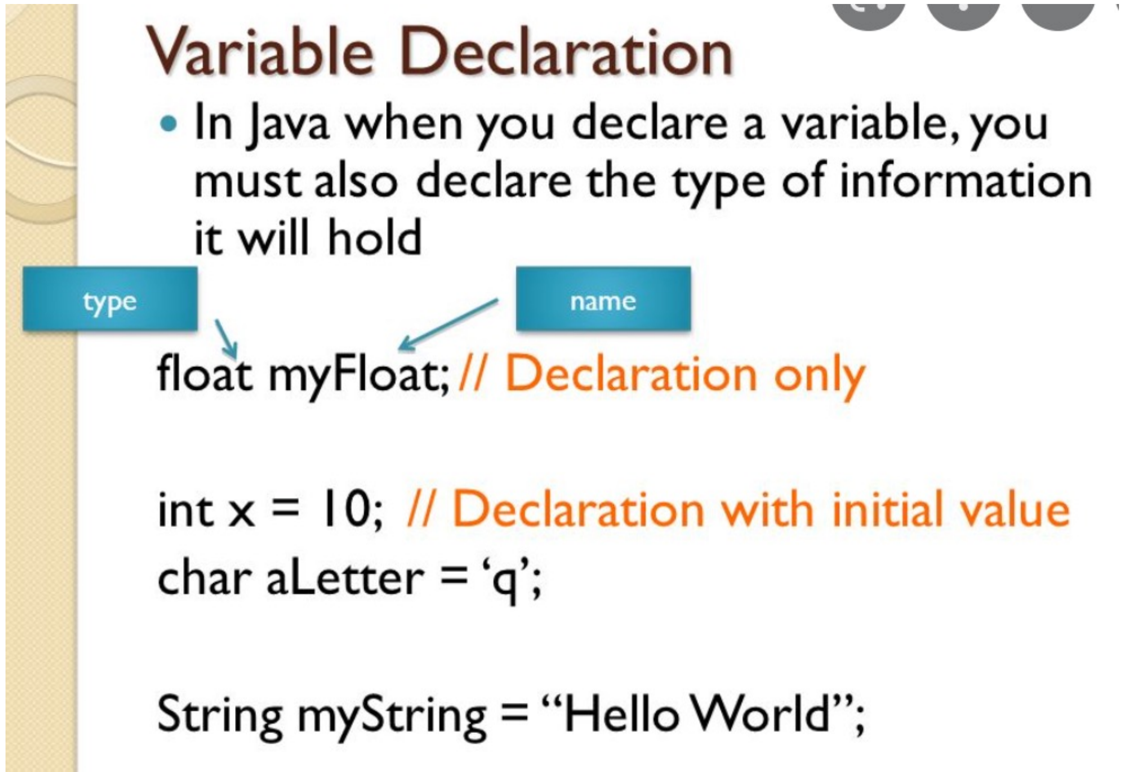
Python: Variable types are dynamically deduced, based on assignments

```
>>> message = 'And now for something completely different'
>>> n = 17
>>> pi = 3.1415926535897931
>>> print(n)
17
>>> print(pi)
3.141592653589793

>>> type(message)
<class 'str'>
>>> type(n)
<class 'int'>
>>> type(pi)
<class 'float'>
```

Different in java and C

More rigid, less error-prone



Variable Declaration

- In Java when you declare a variable, you must also declare the type of information it will hold

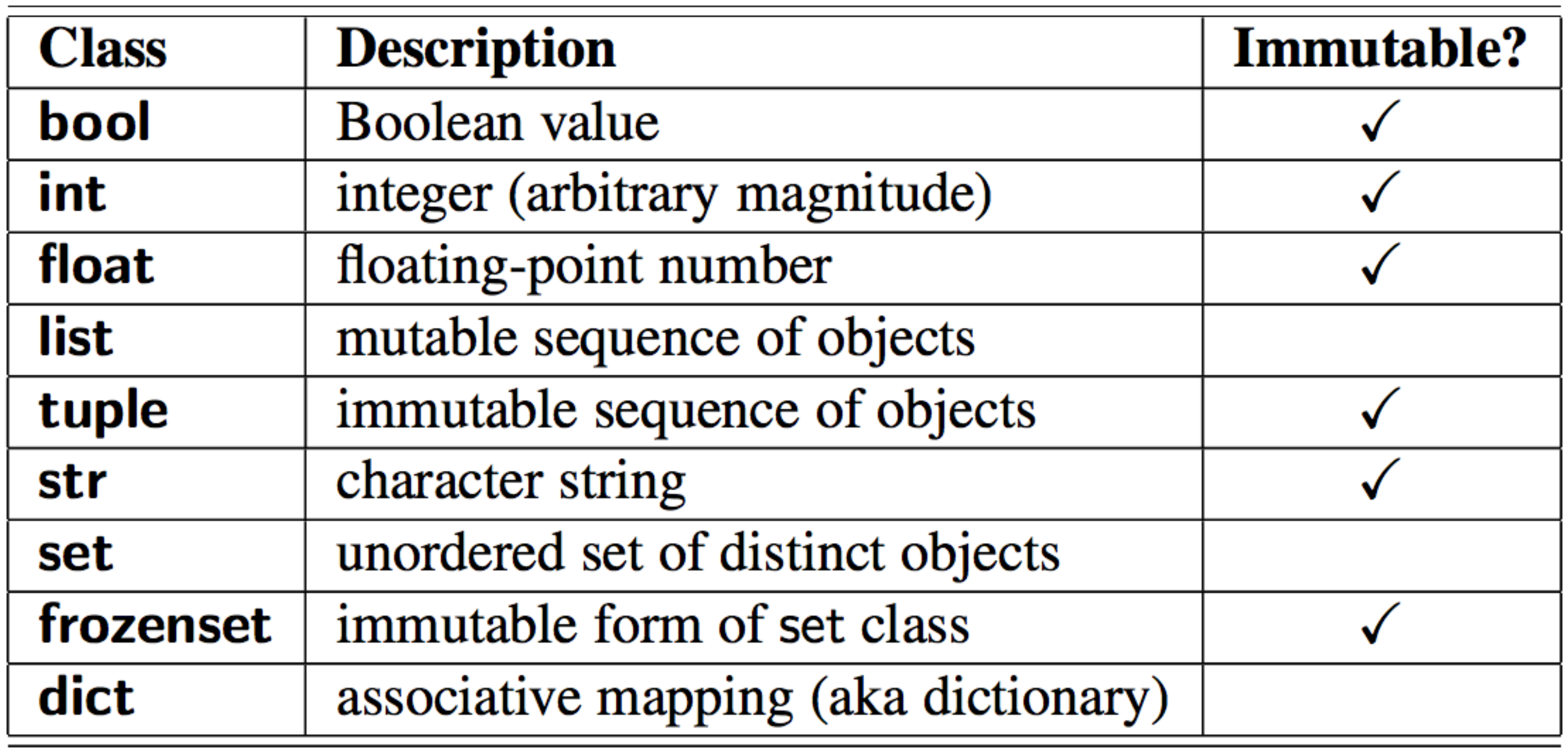
```
float myFloat; // Declaration only

int x = 10; // Declaration with initial value
char aLetter = 'q';

String myString = "Hello World";
```

Built-in Classes

- Immutable: object value cannot be changed
- Identifier can be reassigned



Class	Description	Immutable?
bool	Boolean value	✓
int	integer (arbitrary magnitude)	✓
float	floating-point number	✓
list	mutable sequence of objects	
tuple	immutable sequence of objects	✓
str	character string	✓
set	unordered set of distinct objects	
frozenset	immutable form of set class	✓
dict	associative mapping (aka dictionary)	

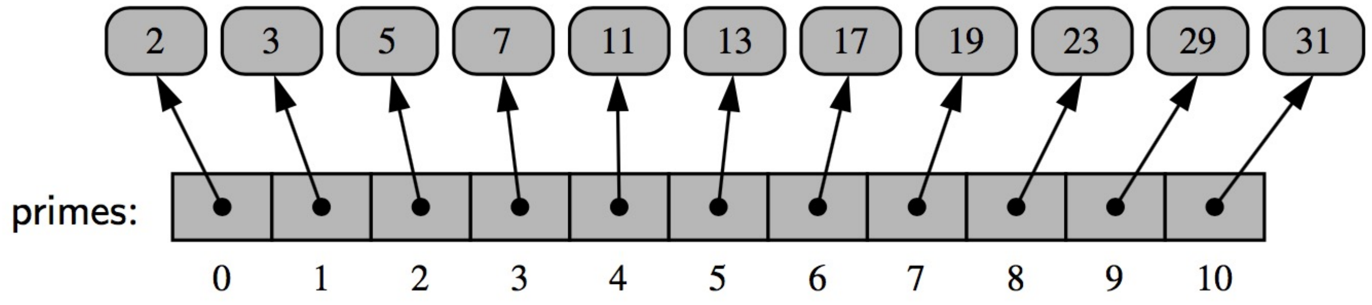
Table 1.2: Commonly used built-in classes for Python

Built-in Classes (Cont’d)

- bool:
  - Values: True or False;   bool() returns False;   bool(val), with val from other types
- int (automatically choose internal representation based on magnitude):
  - initiate by int values: myInt = 10
  - Binary, octal and hexadecimal: 0b1011, 0o52, 0x7f (base of 2, 8 and 16)
  - int() returns 0
  - int(val) return truncated value for val being float: int(3.4), int(3.99), int(-3.9)
  - int(‘137’) converts a string to int if possible
  - int(‘7f’, 16) converts from a different base to decimal
  - Decimal to other bases: bin(…), oct(…), hex(…)
- float (close to double in Java and C/C++):
  - myFloat = 8.9   myFloat = 8.   myFloat = .8   myFloat = 6.022e23   myFloat = float()
  - myFloat = float(2)   myFloat = float(‘3.14’)
  - sys.float_info
- type(…) – check the type of a variable

Built-in Classes (Cont'd)

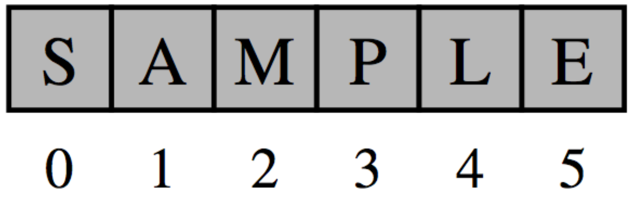
- Sequence classes (list, tuple, str): A collection of values with (important) ordering
- list (mutable)
  - referential: stores an array of references to objects



  - Zero-Indexed: primes[0], ..., primes[len(primes)-1]
  - Can be a mixture of (arbitrary) types, init using values/references
    v_str = ‘tmp str’; v_float = 3.14
    myList = [3, v_str, v_float, ’tmp str again’]
  - Init using an empty list: myList = []
  - Issue: list of lists, aliases for entries, be careful! Not an issue for basic types.
- Disclaimer: when copying code from this slide, the “ ’ ” symbol can be problematic.

Built-in Classes (Cont’d)

- tuple:
  - Immutable version of list, use ‘()’ instead of ‘[]’
  - Once initialized, cannot change values
  - For single element tuple, use myTuple = (17,) instead of myTuple = (17), why?
- str (also immutable):
  - myStr = ’sample’ or “sample”
  - myStr = ”Don’t worry” or ‘Don\’t worry’
  - myStr = ‘C:\\Python\\’   other special chars: ’\n’ – line break; ‘\t’ – tab
  - Use ’’’ or ””” to begin/end a a string literally.



```
print("""Welcome to the GPA calculator.
Please enter all your letter grades, one per line.
Enter a blank line to designate the end.""")
```

Built-in Classes (Cont'd)

- set:
  - A set of elements without ordering (no repeating elements)
  - Implemented using hash table (will talk in the future)
  - Only contains immutables as values (no sets or lists as values)
  - Immutable version – frozenset
  - Use curly braces ' { } '
  - Empty set: use set(), not {} – reserved for empty dictionary
  - { 'red' , 'green' , 'blue' }
  - Constructor: convert an iterable input into a set of its element
    mySet = set('hello') is equivalent to mySet = { 'h' , 'e' , 'l', 'o' }.
    Why one less 'l'?

Built-in Classes (Cont'd)

- dict (dictionary):
  - Mappings from keys to values
    myDict = { ‘ga’ : ‘Irish’ , ‘de’ : ‘German’ }
    pairs = [(‘ga’ , ‘Irish’), (‘de’ , ‘German’)]; myDict = dict(pairs)
    myDict = { }   # empty dictionary

Statements

- A statement: a unit of code that the Python interpreter can execute
- So far: print, assignment
- More to come
- Interactive mode: execute statements one by one
- Script/code: execute sequentially
- Change of order of execution – conditional, loop, etc.

Operators

Operators are special symbols that represent computations like addition and multiplication. The values the operator is applied to are called *operands*.

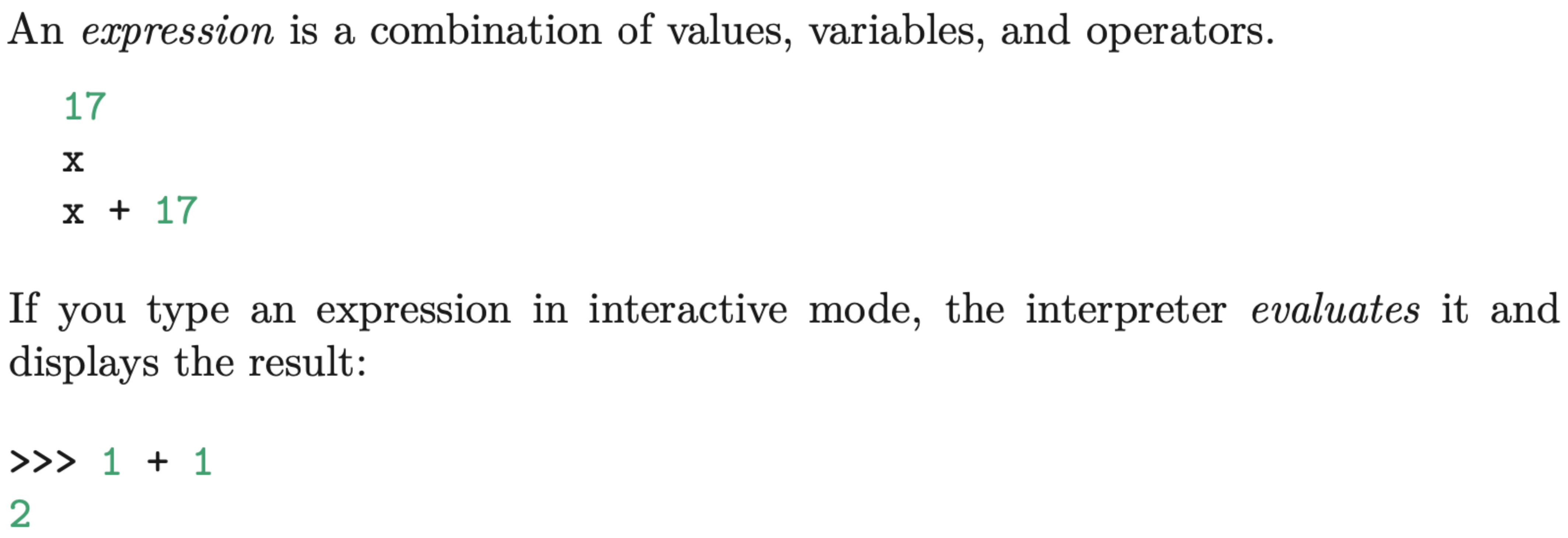
The operators `+`, `-`, `*`, `/`, and `**` perform addition, subtraction, multiplication, division, and exponentiation, as in the following examples:

```
20+32    hour-1    hour*60+minute    minute/60    5**2    (5+9)*(15-7)
```

```
>>> minute = 59
>>> minute/60
0.98333333333333
```

```
>>> minute = 59
>>> minute//60
0
```

Expressions



An *expression* is a combination of values, variables, and operators.

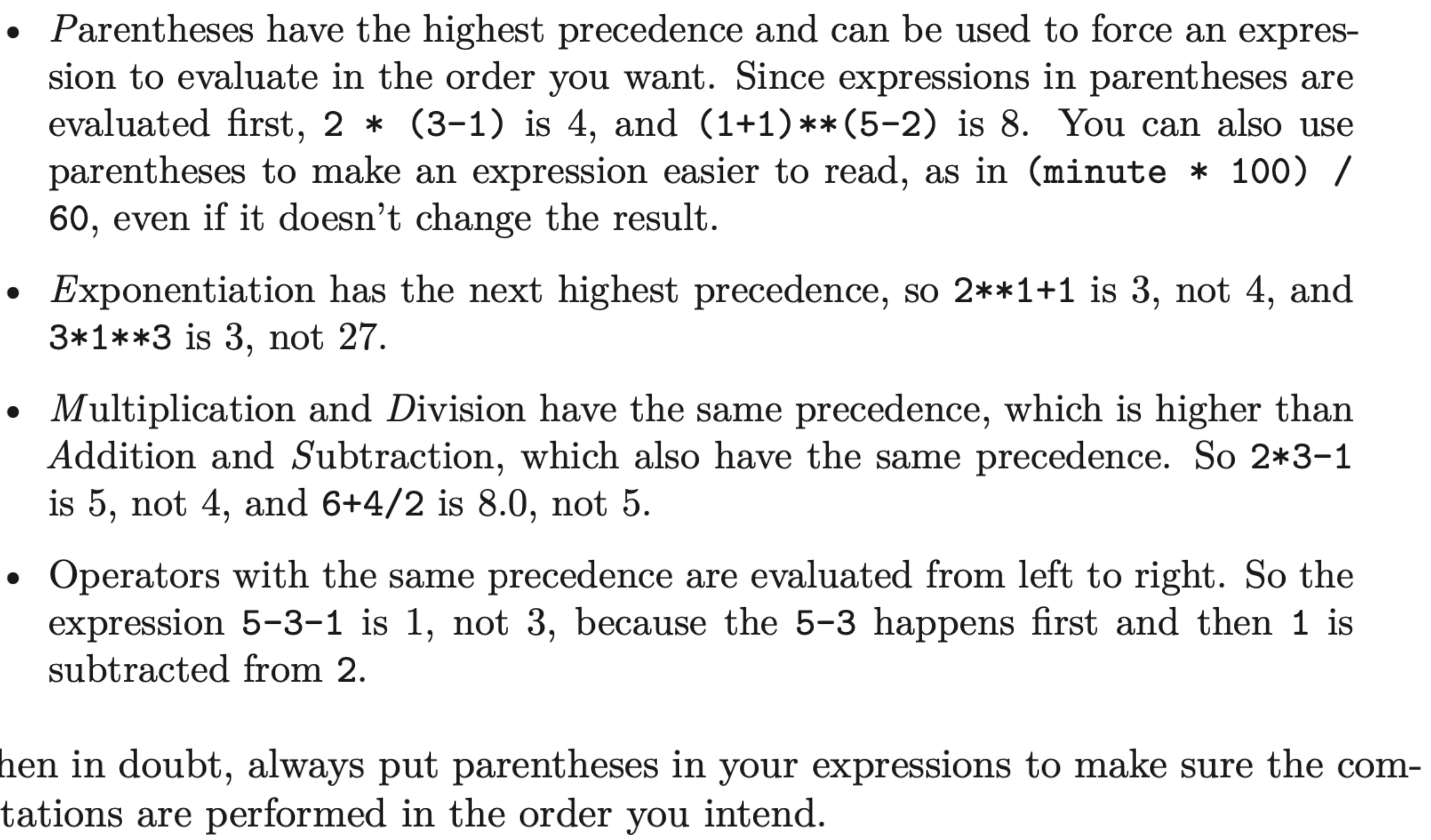
```
17
x
x + 17
```

If you type an expression in interactive mode, the interpreter *evaluates* it and displays the result:

```
>>> 1 + 1
2
```

Order of operations

- Follows mathematical convention -- PEMDAS



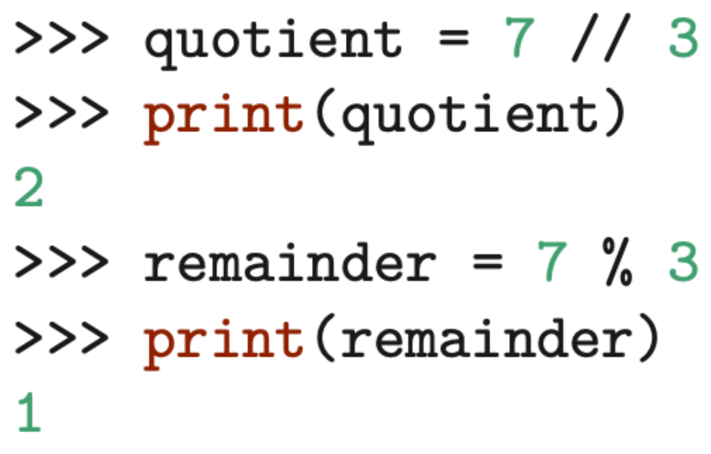
- *P*arentheses have the highest precedence and can be used to force an expression to evaluate in the order you want. Since expressions in parentheses are evaluated first, `2 * (3-1)` is 4, and `(1+1)**(5-2)` is 8. You can also use parentheses to make an expression easier to read, as in `(minute * 100) / 60`, even if it doesn't change the result.
- *E*xponentiation has the next highest precedence, so `2**1+1` is 3, not 4, and `3*1**3` is 3, not 27.
- *M*ultiplication and *D*ivision have the same precedence, which is higher than *A*ddition and *S*ubtraction, which also have the same precedence. So `2*3-1` is 5, not 4, and `6+4/2` is 8.0, not 5.
- Operators with the same precedence are evaluated from left to right. So the expression `5-3-1` is 1, not 3, because the `5-3` happens first and then `1` is subtracted from `2`.

When in doubt, always put parentheses in your expressions to make sure the computations are performed in the order you intend.

What are the values of

- (1+1)**(5-2)
- (1+1)**5-2
- 1+1**(5-2)

Modulus Operator



```
>>> quotient = 7 // 3
>>> print(quotient)
2
>>> remainder = 7 % 3
>>> print(remainder)
1
```

- 7/3 = ?
- Check if x is even or odd?
- The last two digits of x?

- +, -, * → with any float operand, output float
- 27 / 4 = 6.75;   27 // 4 = 6;   27 % 4 = 3
- n = q * m + r   q = n // m;   r = n % m
  m and r always have the same sign, |r| < |m|
- -27 // 4 = -7,   -27 % 4 = 1
- 27 // -4 = -7,   27 % -4 = -1
- Even for floats: 8.2 // 3.14 = 2.0, 8.2 % 3.14 = 1.92

String concatenation – "+"

```
>>> first = 10
>>> second = 15
>>> print(first+second)
25
>>> first = '100'
>>> second = '150'
>>> print(first + second)
100150
```