Valnaos
IT Training & Consulting

ALTRAN
.ALTRAN EDUCATION SERVICES

Python 3 Beginner

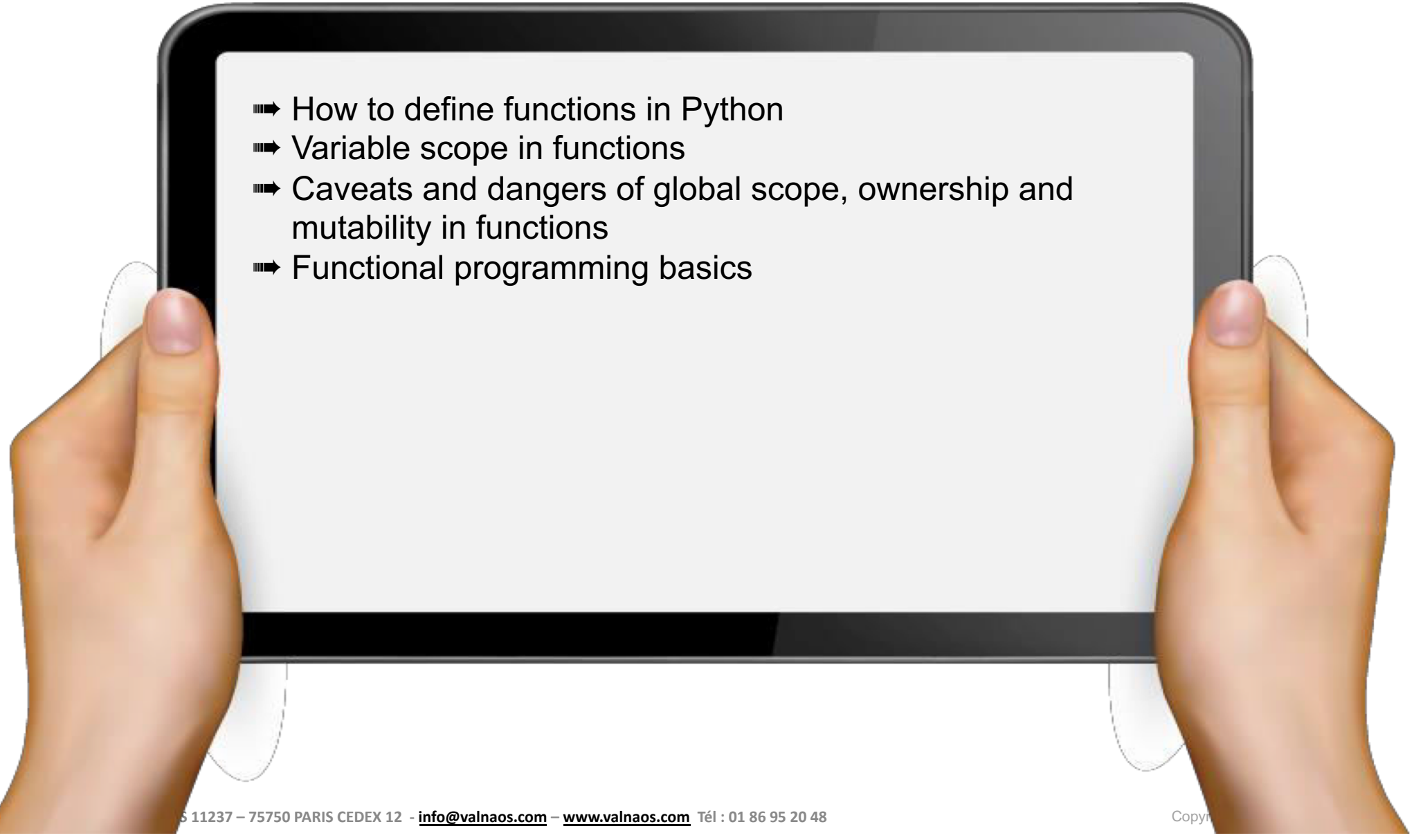
Chapter **00**

Previously, on Python 3 Beginner…

release 1.0.0

37/39 Avenue Ledru Rollin – CS 11237 – 75750 PARIS CEDEX 12 - info@valnaos.com – www.valnaos.com Tél : 01 86 95 20 48

What we saw last time



Q&A

Where? Who?

Class Material

➠ GitHub: https://github.com/cstar-industries/python-3-beginner

Instructor

➠ Charles Francoise <charles@cstar.io>

Session Objectives

At the end of this session, you will be able to:

- Perform operations on the filesystem and paths
- Read from and write to files
- Handle base64-encoded data
- Serialize/deserialize python objects to pickle and JSON
- Use math functions and random number generators
- Handle date and time data with ease in Python
- Be curious to learn more about regular expressions

Session Syllabus

Chapter	Subject	Start	End	Total Time (in hours)
00	Previously on Python 3 Beginner…	14:00	14:30	00:30
01	A Guided Tour of the Python Standard Library	14:30	15:15	00:45
	Coffee Break	*15:15*	*15:30*	*00:15*
01	A Guided Tour of the Python Standard Library	15:30	16:15	00:45
	Coffee Break	*16:15*	*16:30*	*00:15*
01	A Guided Tour of the Python Standard Library	16:30	17:15	00:45
	Workshop	17:15	18:00	00:45
	Total			**04:00**

Valnaos
IT Training & Consulting

ALTRAN
.ALTRAN EDUCATION SERVICES

Python 3 Beginner

Chapter **01**

A guided tour of the Python Standard Library

release 1.0.0

Dealing with files: opening

- Open a file using the built-in function open

```
f = open('hello.txt')
print(f.read())
f.close()
```

```
hello
```

- Opening a file can fail: remember to catch errors

```
f = open('hello.txtx')
```

```
FileNotFoundError: [Errno 2] No such file or directory: 'hello.txtx'
```

Dealing with files: opening

- An open file should always be closed

```
f = open('hello.txt')
...
f.close()
```

⚠ Errors can happen here

- Use a `with` statement to make sure file is closed

```
with open('hello.txt') as f:
    data = f.read()
print(f.closed)
```

```
True
```

Dealing with files: opening

- By default, a file is not writable

```
with open('hello.txt') as f:
    f.write('Hello World!')
```

```
UnsupportedOperation: not writable
```

- Use the `mode` argument

```
with open('hello.txt', 'w') as f:
    f.write('Hello World!')
```

"Write mode"

Dealing with files: opening

- There are many **modes** to open a file

Character	Meaning
`r`	open for reading (default)
`w`	open for writing, truncating the file first
`x`	open for exclusive creation, failing if the file already exists
`a`	open for writing, appending to the end of the file if it exists
`b`	binary mode
`t`	text mode (default)
`+`	open for updating (reading and writing)

- Mode characters can be combined:
  - `'r+b'` opens the file for reading and writing in binary mode
- See the documentation to learn more about `open`

Dealing with files: reading and writing

➠ Read from a file with the `read` method

```
print(f.read(3))      # Number of bytes/characters to be read
print(f.read())       # Read to end of file
```

```
hel
lo
```

➠ Write to a file with the `write` method

```
f.write('Hello ')
f.write('World!')
...
print(f.read())
```

```
Hello World!
```

Dealing with files: reading and writing

- Move your cursor with the `seek` method and find out your current position with the `tell` method

```
f.seek(4)
print(f.tell())
print(f.read())
```

```
o
```

- Find out more about files and *file-like objects* in the documentation

Let’s write some code!

```
with open('hello.txt', 'w') as f:
  f.write('Hello World!')

with open('hello.txt') as f:
  print(f.read())
```

```
Hello World!
```

Interacting with the operating system: the os module

```
import os
import os.path
```

- The built-in os module provides a way to call operating system functions from Python
  - Environment variables
  - Filesystem operations
  - Process management
  - Cryptographically-secure random numbers
- ⚠️ Not all functions are portable between OSes (Linux, macOS, BSD, Windows...)
- Find a comprehensive list of the os module in the documentation

Reading environment variables

- You can list all variables defined in your current environment with os.environ

```
os.environ
```

```
environ{'TERM': 'xterm-256color',
        'SHELL': '/bin/bash',
        'TMPDIR': '/var/folders/_7/5ywmfgxn1v50w94f4bp5ql080000gn/T/',
        'LC_ALL': 'en_US.UTF-8',
        'USER': 'chrales',
        ...
```

- Get the value of a single environment variable or a default value if it is undefined

```
os.getenv('USER', default='python')
```

```
'chrales'
```

Filesystem operations

- List the contents of a directory

```
os.listdir()
```

```
['001-First-Steps', '004-Functions-And-More', '003-Data-Structures', 'README.md',
'002-Control-Flow']
```

```
os.listdir('/bin')
```

```
['cat', 'echo', 'launchctl', 'df', 'pwd', 'test', 'csh', 'wait4path', 'unlink'...
```

- Change the current directory

```
os.chdir('001-First-Steps')
```

Filesystem operations

Create a directory

```
os.mkdir('new-directory')
```

Remove a directory

```
os.rmdir('new-directory')
```

Rename (move) a file or directory

```
os.rename('hello.txt', 'hello-world.txt')
```

```
os.rename('hello.txt', '005-Python-Standard-Library/hello.txt')
```

Remove a file

```
os.remove('005-Python-Standard-Library/hello.txt')
```

Filesystem operations

- Create a symbolic link (Windows "shortcut")

```
os.symlink('Shortcut to classes', '/home/chrales/Classes/python-3-beginner')
```

- Reading a symbolic link

```
os.readlink('Shortcut to classes')
```

```
['cat', 'echo', 'launchctl', 'df', 'pwd', 'test', 'csh', 'wait4path', 'unlink'...
```

Filesystem operations

➠ Query info about paths

```
os.path.exists('hello.txt')
```

```
True
```

```
print(os.path.isfile('hello.txt'))
print(os.path.isdir('hello.txt'))
```

```
True
False
```

```
os.path.getsize('hello.txt')
```

```
13
```

Filesystem operations

➠ Extract path components

```
os.path.split('/home/chrales/Work/python-3-beginner/README.md')
```

```
('/home/chrales/Work/python-3-beginner', 'README.md')
```

```
print(os.path.dirname('/home/chrales/Work/python-3-beginner/README.md'))
print(os.path.basename('/home/chrales/Work/python-3-beginner/README.md'))
```

```
/home/chrales/Work/python-3-beginner
README.md
```

```
os.path.splitext('README.md')
```

```
('hello', '.txt')
```

Filesystem operations

- Join path components

```
os.path.join('/home/chrales', 'Work', 'python-3-beginner', 'README.md')
```

```
'/home/chrales/Work/python-3-beginner/README.md'
```

- Get the absolute canonical path (resolve symlinks, relative paths)

```
os.path.realpath('../python-3-beginner/README.md')
```

```
'/home/chrales/Work/python-3-beginner/README.md'
```

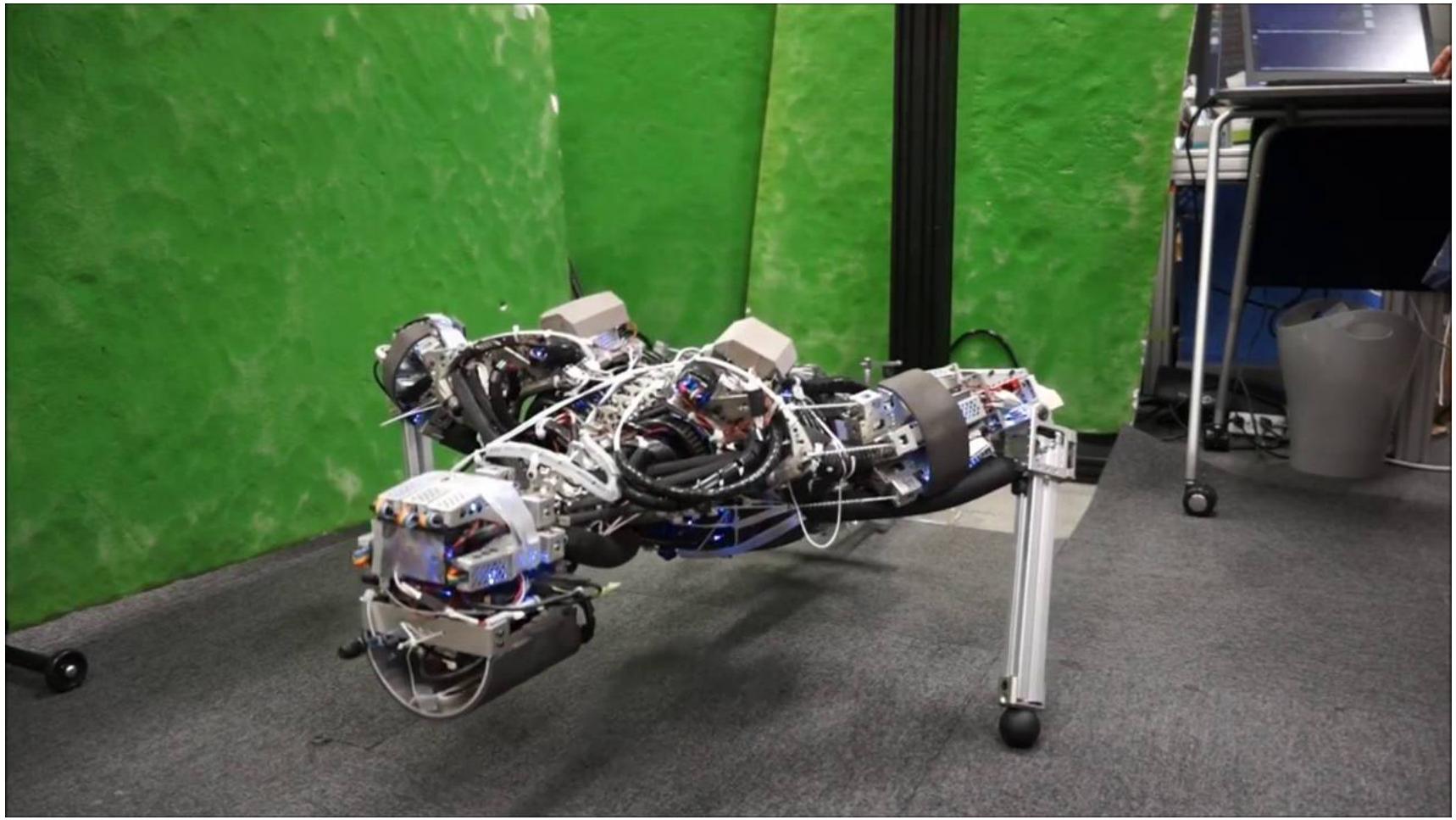
Let's write some code!

```
with open('hello.txt', 'w') as f:
  f.write('Hello World!')

with open('hello.txt') as f:
  print(f.read())
```

```
Hello World!
```

Workout Time!



Regular expressions: the `re` module

```
import re
```

- The built-in `re` module provides regular expression matching operations
- Find out more about the `re` module in the documentation
- Learn about regular expressions at https://regexone.com
- Play regular expressions crosswords at https://regexcrossword.com
- Test and debug your regular expressions at https://regex101.com

Regular expressions: the re module

Match a regex in a string

```
m = re.match(r'a+b', 'aaaab')
print(m)
```

```
<re.Match object; span=(0, 5), match='aaaab'>
```

```
m = re.match(r'a+b', 'baba')
print(m)
```

```
None
```

```
m = re.search(r'a+b', 'baba')
print(m)
```

```
<re.Match object; span=(1, 3), match='ab'>
```

Regular expressions: the re module

- Replace ("substitute") occurrences of a regex

```
re.sub(r'a+', 'a', 'aaaaa-aaa-aaaaaa-a-bbb')
```

```
'a-a-a-a-bbb'
```

```
re.sub(r'\D', '', '(460)190-2235')
```

```
'4601902235'
```

- Split a string using a regular expression

```
re.split(r'[-_/]', 'hello-world/python_rocks')
```

```
['hello', 'world', 'python', 'rocks']
```

Encoding binary data to text with `base64`

- Base-64 is a text-only encoding that can be used to encode any binary data to 64 "safe" characters:
  - A-Z uppercase: 26
  - a-z lowercase: 26
  - 0-9 digits: 10
  - + and /: 2
- This encoding is useful for sending binary data over the network (e.g. by email) where unexpected character re-interpretation can occur
- Often used to transfer cryptographic keys and certificates

Encoding and decoding with base64

- Encode binary data to base64

```
base64.b64encode(b'Hello World')
```

```
b'SGVsbG8gV29ybGQh'
```

- Encode base64 to binary data

```
base64.b64decode(b'SGVsbG8gV29ybGQh')
```

```
b'Hello World!'
```

Math functions with the `math` module

```
import math
```

➠ Math constants

```
math.pi
```

```
3.141592653589793
```

```
math.e
```

```
2.718281828459045
```

```
math.tau
```

```
6.283185307179586
```

Math functions with the `math` module

➠ Sentinel values

```
math.nan
```

```
nan
```

```
math.inf
```

```
inf
```

```
-math.inf
```

```
-inf
```

Math functions with the `math` module

- Convert from degrees to radians

```
math.radians(180)
```

```
3.141592653589793
```

```
math.degrees(math.pi / 4)
```

```
45.0
```

- Use trigonometric functions

```
math.cos(math.pi / 3)
```

```
0.5
```

Math functions with the `math` module

- More math functions

```
math.log(2)          # natural logarithm
math.log10(100)      # base 10 logarithm
math.log2(1024)      # base 2 logarithm

math.acos(0.707)     # Inverse cosine
math.tanh(1)         # Hyperbolic tangent

math.gcd(16, 642)    # Greatest common divisor
math.hypot(3, 4)     # Hypothenuse (Pythagoras theorem)
```

- Read the list of math functions in the documentation

Random functions

```
import random
```

➠ Get a random number between 0 and 1

```
random.random()
```

```
0.0746095954672188
```

➠ Get a random integer between 10 and 100

```
random.randrange(10, 100)
```

```
59
```

Random functions

➠ Get a random item from a sequence

```
random.choice(['banana', 'pear', 'orange', 'peach'])
```

```
'pear'
```

➠ Pick a random item 5 times

```
random.choices(['banana', 'pear', 'orange', 'peach'], k=5)
```

```
['banana', 'orange', 'banana', 'peach', 'orange']
```

➠ Pick 2 distinct items randomly

```
random.sample(['banana', 'pear', 'orange', 'peach'], k=2)
```

```
['peach', 'banana']
```

Random functions

➠ Get a random item from a sequence

```
random.choice(['banana', 'pear', 'orange', 'peach'])
```

```
'pear'
```

➠ Pick a random item 5 times

```
random.choices(['banana', 'pear', 'orange', 'peach'], k=5)
```

```
['banana', 'orange', 'banana', 'peach', 'orange']
```

➠ Pick 2 distinct items randomly

```
random.sample(['banana', 'pear', 'orange', 'peach'], k=2)
```

```
['peach', 'banana']
```

Random functions

- Various random distributions

```
random.uniform(10, 100)
```

```
80.58706066003606
```

```
random.gauss(0.0, 1.0)
```

```
-0.4858548215268525
```

```
random.betavariate(2.0, 2.0)
```

```
0.8849856235679385
```

Managing time: the `datetime` module

- Time and date is a *hard* subject in programming. Python makes it easy for us.
- Well... easi*er* (documentation)

```
from datetime import datetime, timedelta, timezone
```

- Get the current time and date as a `datetime` object

```
t = datetime.now()
t
```

```
datetime.datetime(2020, 4, 22, 22, 54, 13, 837398)
```

- datetime(year, month, day, hours, minutes, seconds, nanoseconds)

Managing time: the `datetime` module

➠ Create an arbitrary `datetime` object

```
t2 = datetime(year=1984, month=1, day=24)
print(t2)
```

```
1984-01-24 00:00:00
```

➠ Compare `datetimes`

```
print(t < t2)    # t happens before t2
print(t > t2)    # t happens after t2
```

```
False
True
```

Managing time: the `datetime` module

- Get the duration between two `datetimes`

```
t - t2
```

```
datetime.timedelta(days=13238, seconds=83603, microseconds=66028)
```

- Create an arbitrary `timedelta`

```
d = timedelta(days=1, hours=5)
```

- Perform time calculations

```
t - d        # 1 day and 5 hours before
```

```
datetime.datetime(2020, 4, 21, 18, 13, 23, 66028)
```

Managing time: the `datetime` module

➠ Create an arbitrary `datetime` object

```
t2 = datetime(year=1984, month=1, day=24)
print(t2)
```

```
1984-01-24 00:00:00
```

➠ Compare `datetimes`

```
print(t < t2)    # t happens before t2
print(t > t2)    # t happens after t2
```

```
False
True
```

Managing time: the `datetime` module

- Get a `float` timestamp, the # of seconds since Jan 1, 1970 00:00 UTC

```
t.timestamp()
```

```
1587590547.289774
```

- Get a `datetime` from a timestamp

```
datetime.fromtimestamp(0.0)
```

```
datetime.datetime(1970, 1, 1, 1, 0)
```

Managing time: the `datetime` module

➠ Output a `datetime` in ISO 8601 format

```
t.isoformat()
```

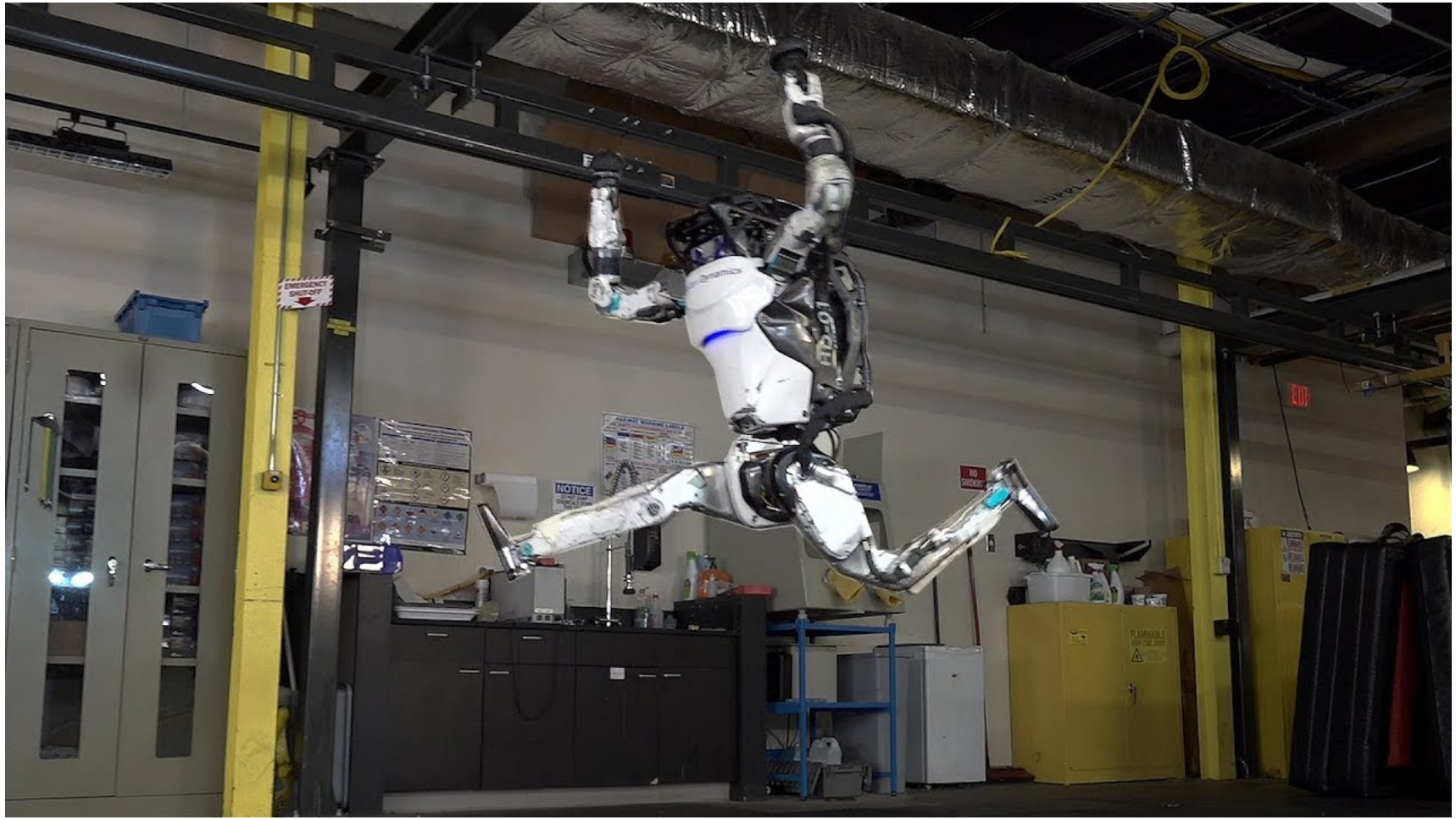
```
'2020-04-22T23:13:23.066028'
```

➠ Output a `datetime` to a custom format

```
t.strftime('%a, %b %d %H:%m %Y')
```

```
'Wed, Apr 22 23:04 2020'
```

Workout Time!



Serializing Python data: the `pickle` module

```
import pickle
```

➠ Save a Python object to a file

```
d = {'date': datetime.now(), 'temperature': 19.7, 'python': 'snek'}
with open('python_obj.pickle', 'wb') as f:
    pickle.dump(d, f)
```

```
00000000  80 03 7d 71 00 28 58 04  00 00 00 64 61 74 65 71  |..}q.(X....dateq|
00000010  01 63 64 61 74 65 74 69  6d 65 0a 64 61 74 65 74  |.cdatetime.datet|
00000020  69 6d 65 0a 71 02 43 0a  07 e4 04 16 17 38 14 0c  |ime.q.C......8..|
00000030  de ae 71 03 85 71 04 52  71 05 58 0b 00 00 00 74  |..q..q.Rq.X....t|
00000040  65 6d 70 65 72 61 74 75  72 65 71 06 47 40 33 b3  |emperatureq.G@3.|
00000050  33 33 33 33 33 58 06 00  00 00 70 79 74 68 6f 6e  |33333X....python|
00000060  71 07 58 04 00 00 00 73  6e 65 6b 71 08 75 2e     |q.X....snekq.u.|
```

Serializing Python data: the `pickle` module

➠ Reading a Python object back from a pickle file

```
with open('python_obj.pickle', 'rb') as f:
    d2 = pickle.load(f)
print(d2)
```

```
{'date': datetime.datetime(2020, 4, 22, 21, 59, 41, 32105),
'temperature': 19.7, 'python': 'snek'}
```

```
d2 == d
```

```
True
```

Serializing Python data: the `json` module

```
import json
```

- JSON is a text-based, human-readable, interchange format based on JavaScript
- It is ubiquitous as a data format, particularly from web services
- JSON is common in many programming languages, making a good interchange format between programs
- ⚠️ Not all Python object can be serialized to or deserialized from JSON

Serializing Python data: the `json` module

➠ Reading data from a JSON string

```
json_data = '{"first_name": "Rob", "last_name": "Pike", "occupation":
"programmer"}'
d = json.loads(json_data)
print(d)
```

```
{'first_name': 'Rob', 'last_name': 'Pike', 'occupation': 'programmer'}
```

➠ Writing a Python object to a JSON file

```
with open('python_obj.json', 'w') as f:
    json.dump(d, f)
```

```
00000000  7b 22 66 69 72 73 74 5f  6e 61 6d 65 22 3a 20 22  |{"first_name": "|
00000010  52 6f 62 22 2c 20 22 6c  61 73 74 5f 6e 61 6d 65  |Rob", "last_name|
00000020  22 3a 20 22 50 69 6b 65  22 2c 20 22 6f 63 63 75  |": "Pike", "occu|
00000030  70 61 74 69 6f 6e 22 3a  20 22 70 72 6f 67 72 61  |pation": "progra|
00000040  6d 6d 65 72 22 7d                                 |mmer"}|
```

Serializing Python data: the csv module

```
import csv
```

➠ Reading a CSV file

```
with open('tesla-stock-price.csv', newline='') as f:
    csv_data = csv.reader(f, delimiter=",", quotechar='"')
    for row in csv_data:
        print(row)
```

```
['date', 'close', 'volume', 'open', 'high', 'low']
['11:34', '270.49', '4,787,699', '264.50', '273.88', '262.24']
['2018/10/15', '259.5900', '6189026.0000', '259.0600', '263.2800', '254.5367']
['2018/10/12', '258.7800', '7189257.0000', '261.0000', '261.9900', '252.0100']
['2018/10/11', '252.2300', '8128184.0000', '257.5300', '262.2500', '249.0300']
['2018/10/10', '256.8800', '12781560.0000', '264.6100', '265.5100', '247.7700']
['2018/10/09', '262.8000', '12037780.0000', '255.2500', '266.7700', '253.3000']
...
```

Serializing Python data: the `csv` module

- Writing a CSV file

```
with open('contacts.csv', 'w', newline='') as f:
    csv_file = csv.writer(f)
    csv_file.writerow('First name', 'Last name', 'Phone number')
    csv_file.writerows(contacts)    # contacts is a list of lists
```

- Discover more about `pickle`, `json` and `csv` in the documentation

Manipulating zip files: the `zipfile` module

```
from zipfile import ZipFile
```

➠ Opening a zip file

```
z = ZipFile('archive.zip')
...
z.close()
```

➠ You can use a with statement with `ZipFile`

```
with ZipFile('archive.zip') as z:
    ...
```

Manipulating zip files: the `zipfile` module

- List all files in an archive

```
for filename in z.namelist():
    print(filename)
```

```
photos/
photos/IMG_0689.jpg
photos/IMG_0690.jpg
photos/IMG_0691.jpg
...
```

- Extract all files from an archive

```
z.extractall()                          # Extracts to current directory
z.extractall('/home/chrales/pictures') # Extracts to given directory
```

Manipulating zip files: the `zipfile` module

- Extract a specific file (destination path is optional)

```
z.extract('photos/IMG_0690.jpg', 'IMG_0690.jpg')
```

- Read the contents of a file to `bytes`

```
file_data = z.read('photos/IMG_0690.jpg')
print(file_data)
```

```
b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x01\x01\x00\x00\x01...
```

- Close a zip file

```
z.close()
```

Manipulating zip files: the `zipfile` module

- You can also create zip files and write to them

```
with ZipFile('archive_2.zip', 'w') as z:
    z.write('/home/chrales/Pictures/profile.jpg', 'profile.jpg')
```

- Read and write files from an archive in a `file`-like manner

```
with ZipFile('archive_2.zip', 'w') as z:
    with z.open('python_obj.pickle', 'w'):
        pickle.dump(obj, f)
```

Handling SQLite databases: the `sqlite3` module

```
import sqlite3
```

- SQLite is a powerful file-based relational database
- It can be used to serialize complex data to a file
- SQLite is ubiquitous as a storage format
  - Mobile app data
  - Embedded systems
  - Even in space...!
- Learn more about SQLite at https://www.sqlitetutorial.net/

Handling SQLite databases: the `sqlite3` module

- Open a database file

```
conn = sqlite3.connect('database.db')
```

- Create a "cursor"

```
cur = conn.cursor()
```

- Execute a query and iterate over the results

```
for row in cur.execute('SELECT * FROM artists'):
    print(row)
```

```
(1, 'AC/DC')
(2, 'Accept')
(3, 'Aerosmith')
(4, 'Alanis Morissette')
...
```

Handling SQLite databases: the `sqlite3` module

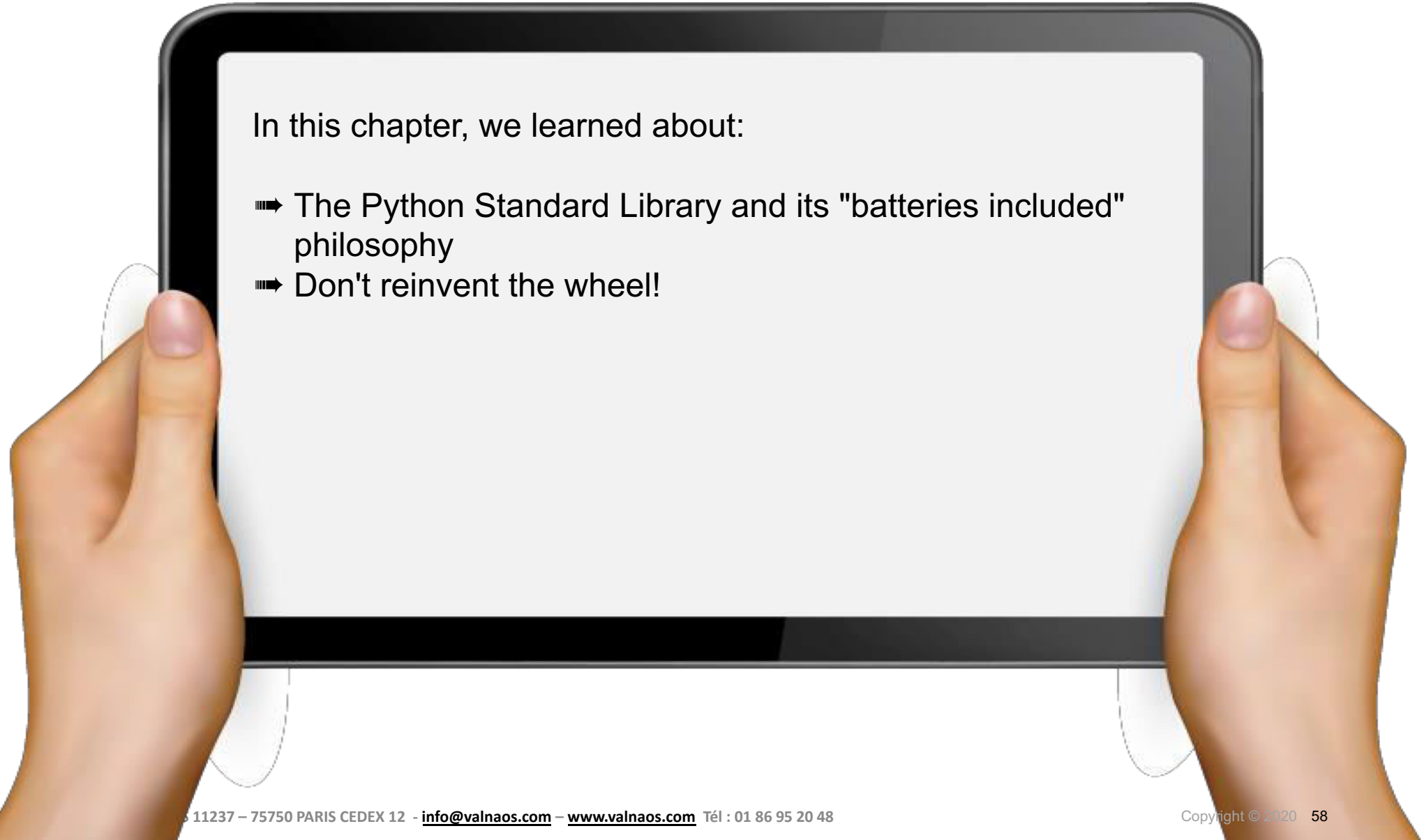
- You can also execute insertions with variable substitution

```
cur.execute('INSERT INTO artists (Name) VALUES (?)', artist_name)
```

- Close the connection and free memory

```
conn.close()
```

Chapter Summary



In this chapter, we learned about:

- The Python Standard Library and its "batteries included" philosophy
- Don't reinvent the wheel!

Q&A

Workshop

Thank you for your attention!

