Valnaos
IT Training & Consulting

ALTRAN
.ALTRAN EDUCATION SERVICES

Python 3 Beginner

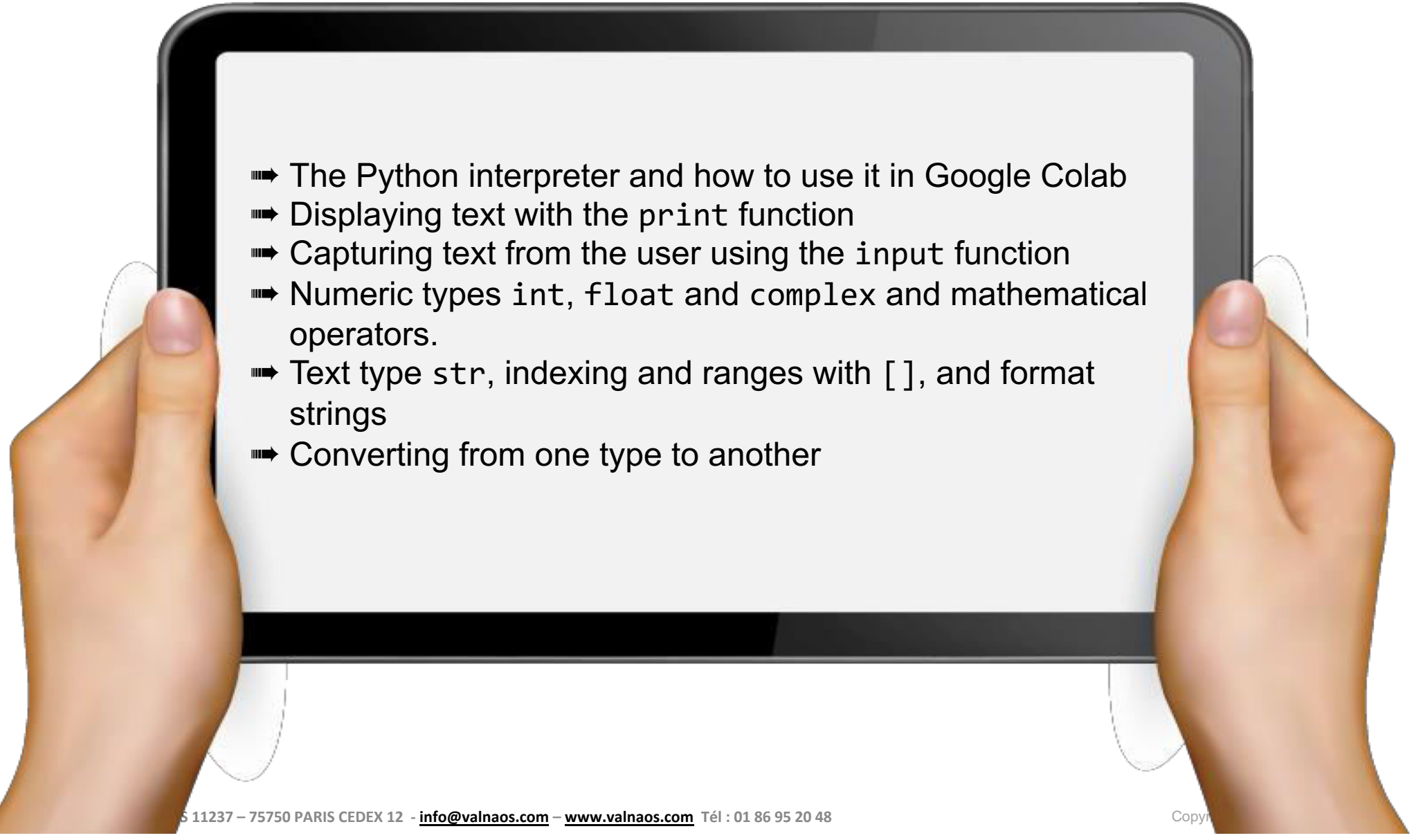
Chapter **00**

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



- The Python interpreter and how to use it in Google Colab
- Displaying text with the `print` function
- Capturing text from the user using the `input` function
- Numeric types `int`, `float` and `complex` and mathematical operators.
- Text type `str`, indexing and ranges with `[]`, and format strings
- Converting from one type to another

S 11237 – 75750 PARIS CEDEX 12 - info@valnaos.com – www.valnaos.com Tél : 01 86 95 20 48

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:

Session Syllabus

Chapter	Subject	Start	End	Total Time (in hours)
00	Previously on Python 3 Beginner…	14:00	14:30	00:30
01	More about `list`	14:30	15:00	00:30
02	Other sequence types: `str` – `tuple` – `set`	15:00	15:45	00:45
	Coffee Break	*15:45*	*16:00*	*00:15*
03	Associative container: `dict`	16:00	16:30	00:30
04	Mastering sequence types	16:30	17:00	00:30
	Coffee Break	*17:00*	*17:15*	*00:15*
	Workshop: mastering control flow	17:15	18:00	00:45
	Total			**04:00**

Valnaos
IT Training & Consulting

ALTRAN
.ALTRAN EDUCATION SERVICES

Python 3 Beginner

Chapter **01**

Boolean type and comparison operators

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

Representing “nothing” in Python

- Special datatype `NoneType`
- Only one value: `None`
- Used to represent the absence of a value (e.g. when a function returns nothing)
- Similar to `null` or `void` in other languages

```
a = None
print(a)
```

```
None
```

Boolean type

- `bool` is a type to contain the Boolean notion of truth
- Two possible values: `True`/`False` (watch out for the capitals!)
- Use the `not` operator to negate the value

	`False`	`True`
`not`	`True`	`False`

Boolean operators

- Boolean values can be combined using Boolean arithmetic operators: `and` / `or`

and	False	True
False	False	False
True	False	True

or	False	True
False	False	True
True	True	True

Boolean operators

- Just like mathematical and string operators, Boolean operators can be prioritized by parentheses
- Watch out for unexpected operator precedence effects! When in doubt, always make your intent explicit with parentheses.

⚠️
```
print(True or True and False)
```
👍
```
print(True or (True and False))
```

```
False
True
```

Let's write some code!

```
print(False or False)
print(False or True)
print(True or False)
print(True or True)
```

```
False
True
True
True
```

Comparison operators

- Comparison operators generate a Boolean value by comparing two values
- Equality: == / Inequality !=
- Greater than: > / Lower than: <
- Greater than or equal: >= / Lower than or equal: <=

```
print(1 == 2)
print(1 < 2)
print('toto' == 'toto')
```

```
False
True
True
```

More comparison operators

- Identity: `is` / `is not`
- Inclusion: `in` / `not in`

```
a = 'Hello World!'
print('x' in a)
print('W' in a)

b = 'Hello World!'
print(a == b)
print(a is b)
```

```
False
True

True
False
```

Complex boolean expressions

➠ Combine comparison results with boolean operators

```
a = 13
(a > 0) and ((a % 2) == 0)
```

```
False
```

➠ Chain comparison operators left to right

➠ `x op1 y op2 z` ⇔ `(x op1 y) and (x op2 z)`

```
a = 13
0 < a < 15
```

```
True
```

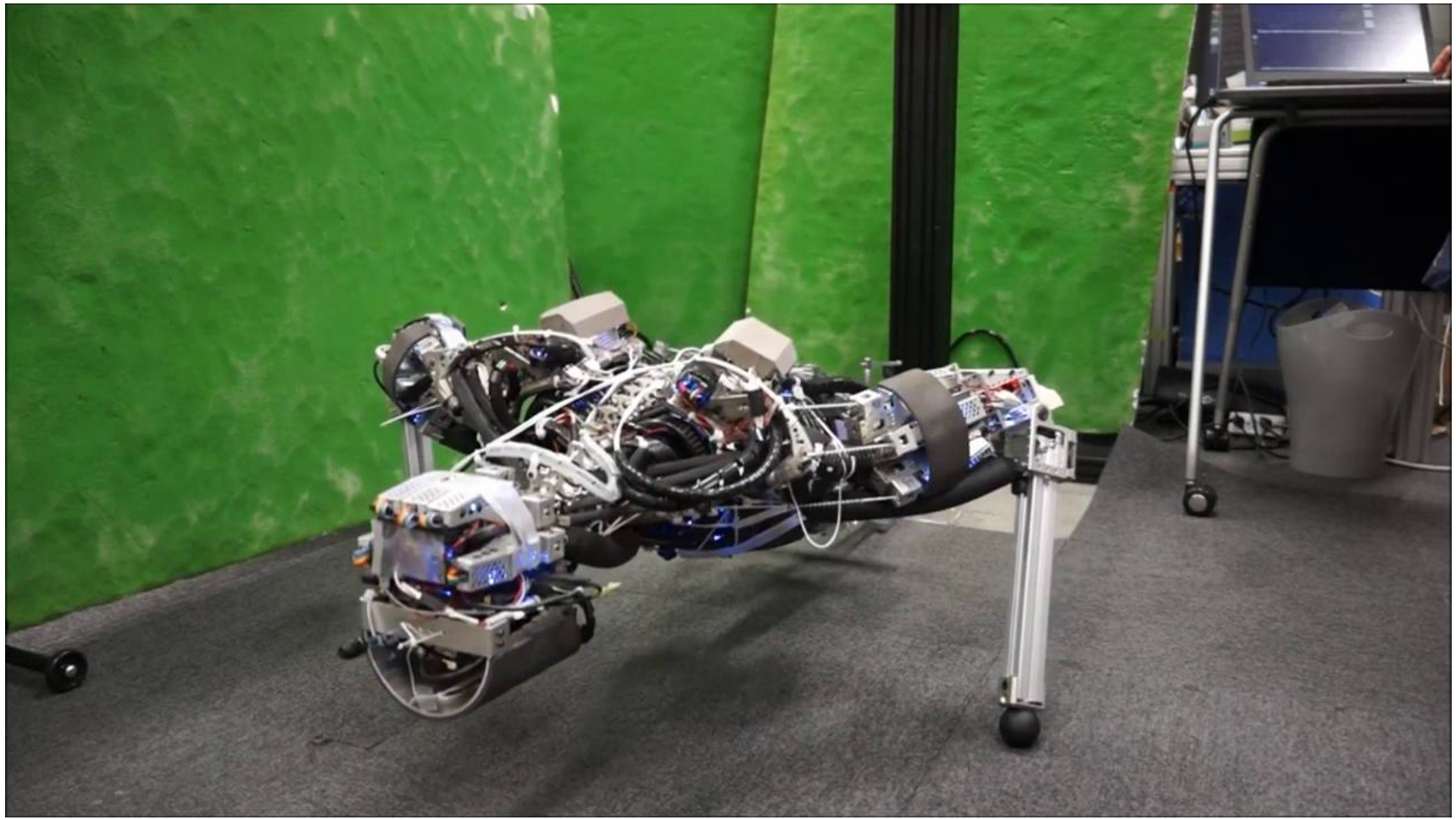
Let's write some code!

```
a = 'Hello World!'
b = 'Hello World!'
print(a == b)

print('h' in a)
print('H' in a)
print('Hell' in a)
```

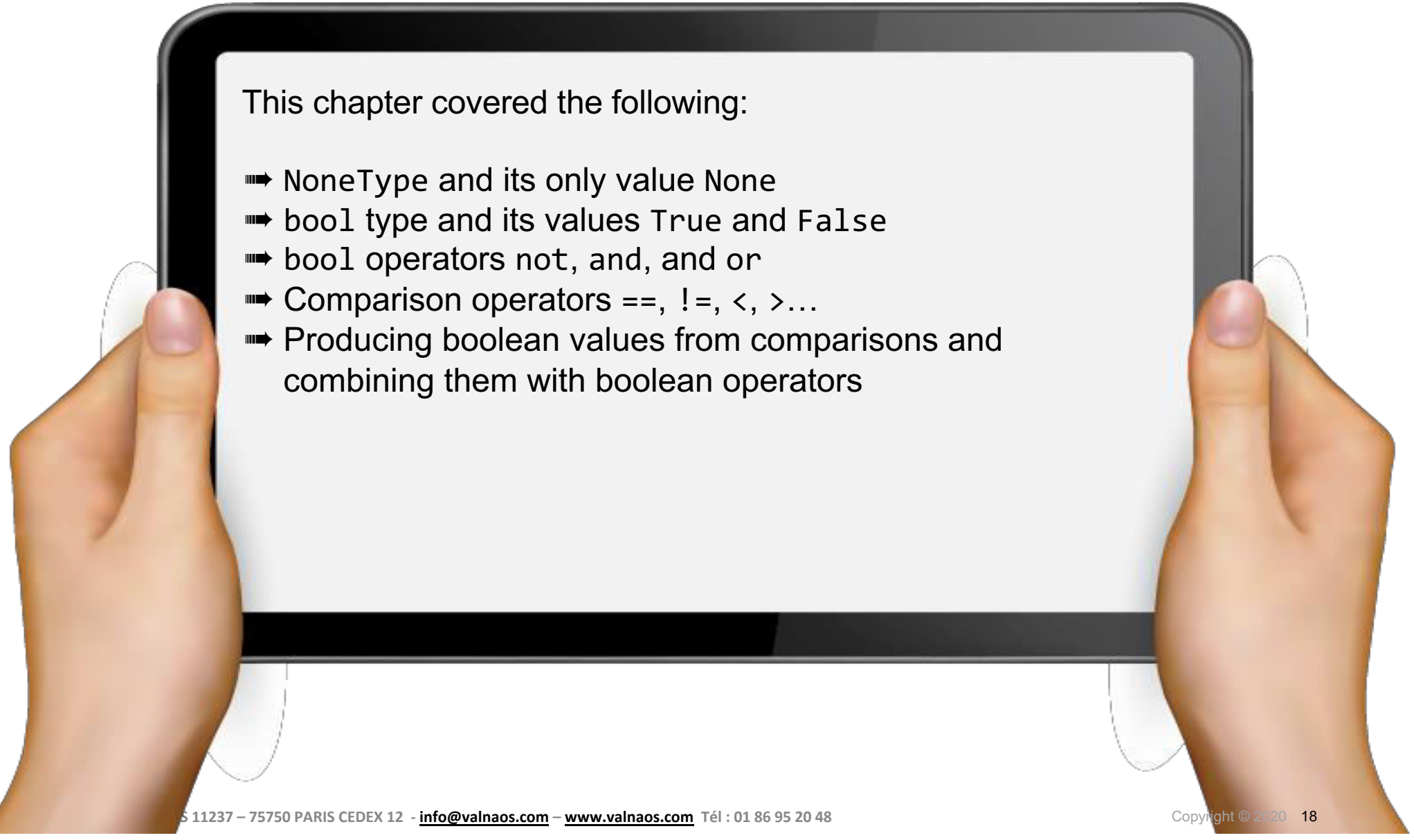
```
True
False
True
True
```

Workout Time!



Chapter 1 Exercises link

Chapter Summary



This chapter covered the following:

- `NoneType` and its only value `None`
- `bool` type and its values `True` and `False`
- `bool` operators `not`, `and`, and `or`
- Comparison operators `==`, `!=`, `<`, `>`…
- Producing boolean values from comparisons and combining them with boolean operators

Valnaos
IT Training & Consulting

ALTRAN
.ALTRAN EDUCATION SERVICES

Python 3 Beginner

Chapter **02**

Conditional Flow: `if` – `else`

release 1.0.0

Conditional flow diagram

➠ How do we display the absolute value of a number?

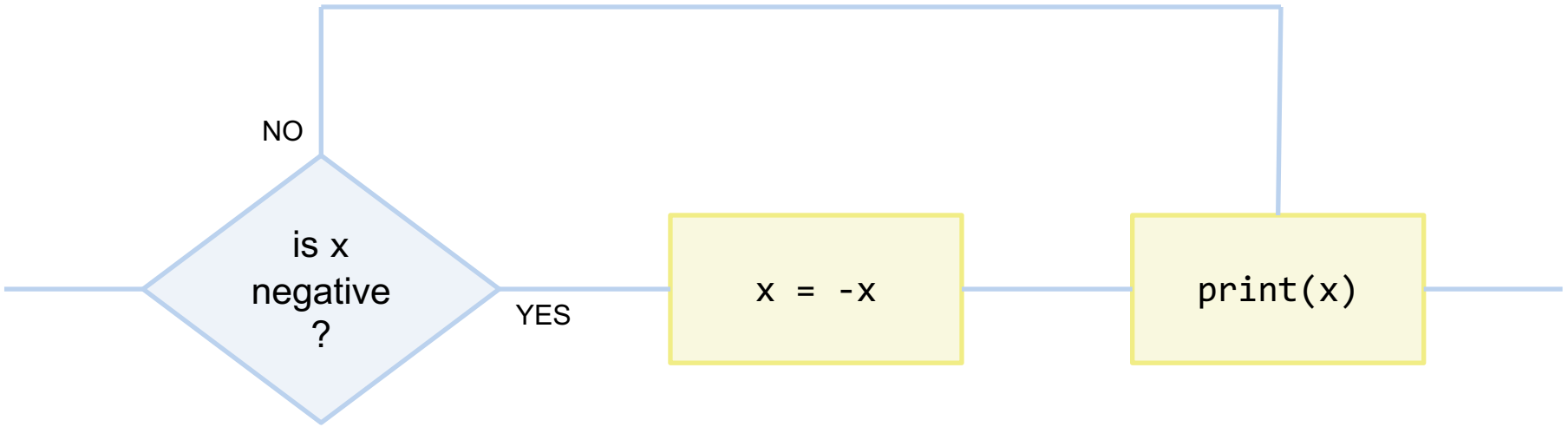
Conditional flow diagram

➠ How do we display the absolute value of a number?

$$|x| = \begin{cases} x, & \text{if } x \geq 0 \\ -x, & \text{if } x < 0. \end{cases}$$

Conditional flow diagram

➠ How do we display the absolute value of a number?



Conditional Flow in Python: `if`

- The `if` keyword either runs or skips code based on a condition

```
x = int(input())

if x < 0:
    x = -x

print(x)
```

Conditional Flow in Python: `if`

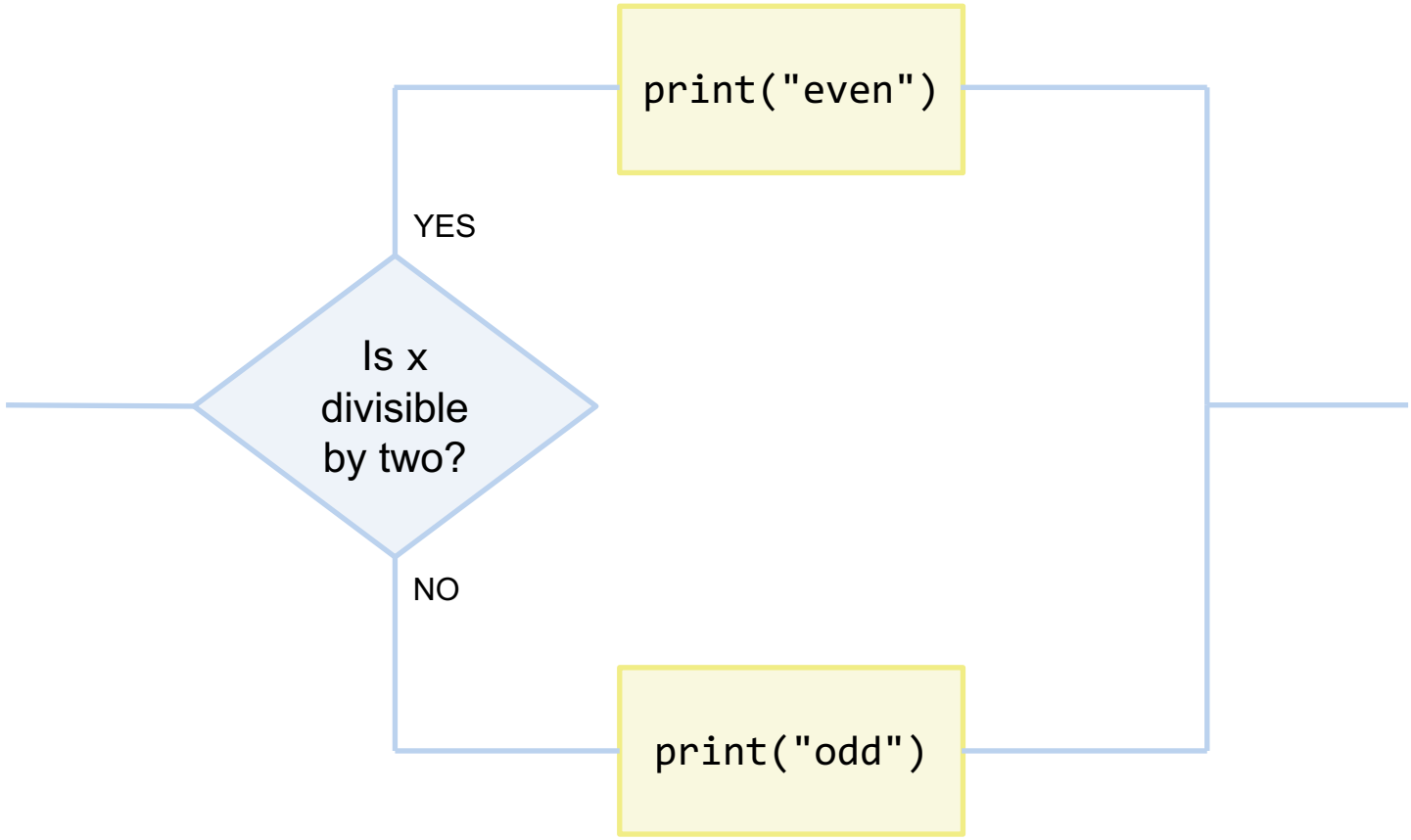
- `if` *`condition`*`:`
- Indentation is important in Python: use 4 spaces

```
x = int(input())

if x < 0:
    x = -x

print(x)
```

Conditional flow diagram



Conditional flow in Python: `else`

- `if` *`condition`*`:`

  `else:`

- Indentation: `else` *must* be aligned with corresponding `if`

```
x = int(input())

if (x % 2) == 0:
    print("even")
else:
    print("odd")
```

Chaining conditions: `elif`

➠ `if` *`condition1`*`:`

`elif` *`condition2`*`:`

`else:`

```
x = int(input())

if x < 0:
    print("negative")
elif (x % 2) == 0:
    print("positive even")
else:
    print("positive odd")
```

What constitutes a *condition*?

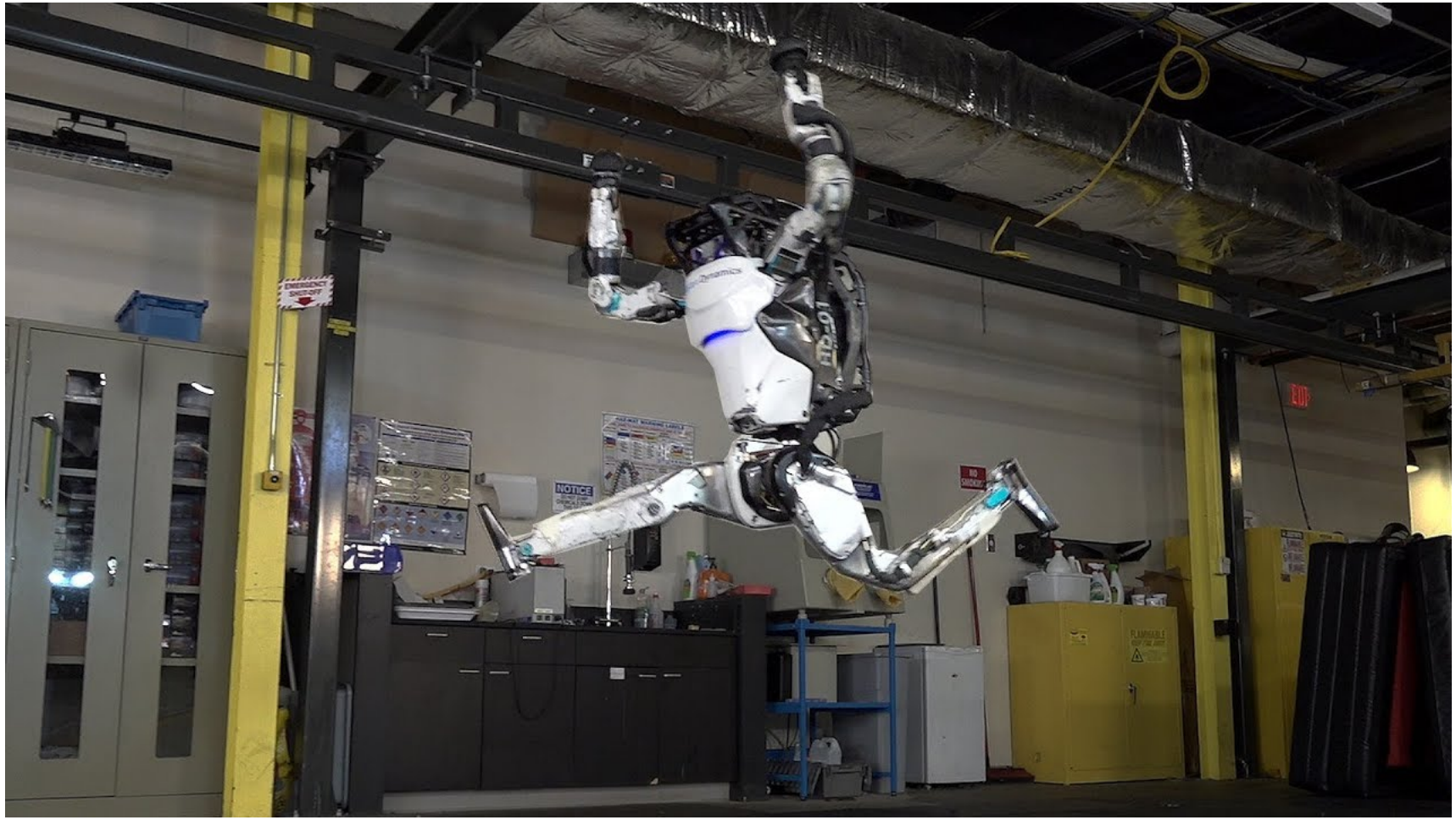
- Any object can be tested for a truth value in an `if` statement.
- By default, an object evaluates to `True`
- The following values evaluate to `False`
  - constants defined to be false: `None` and `False`
  - zero of any numeric type: `0`, `0.0`, `0j`, `Decimal(0)`, `Fraction(0, 1)`
  - empty sequences and collections: `""`, `()`, `[]`, `{}`, `set()`, `range(0)`

Let's write some code!

```
x = 12
if (x % 2) == 0:
    print(f"{x} is even")
    print(f"{x}/2 = {x//2}")
else:
    print(f"{x} is odd")
    print(f"{x}/2 = {x/2}")
```

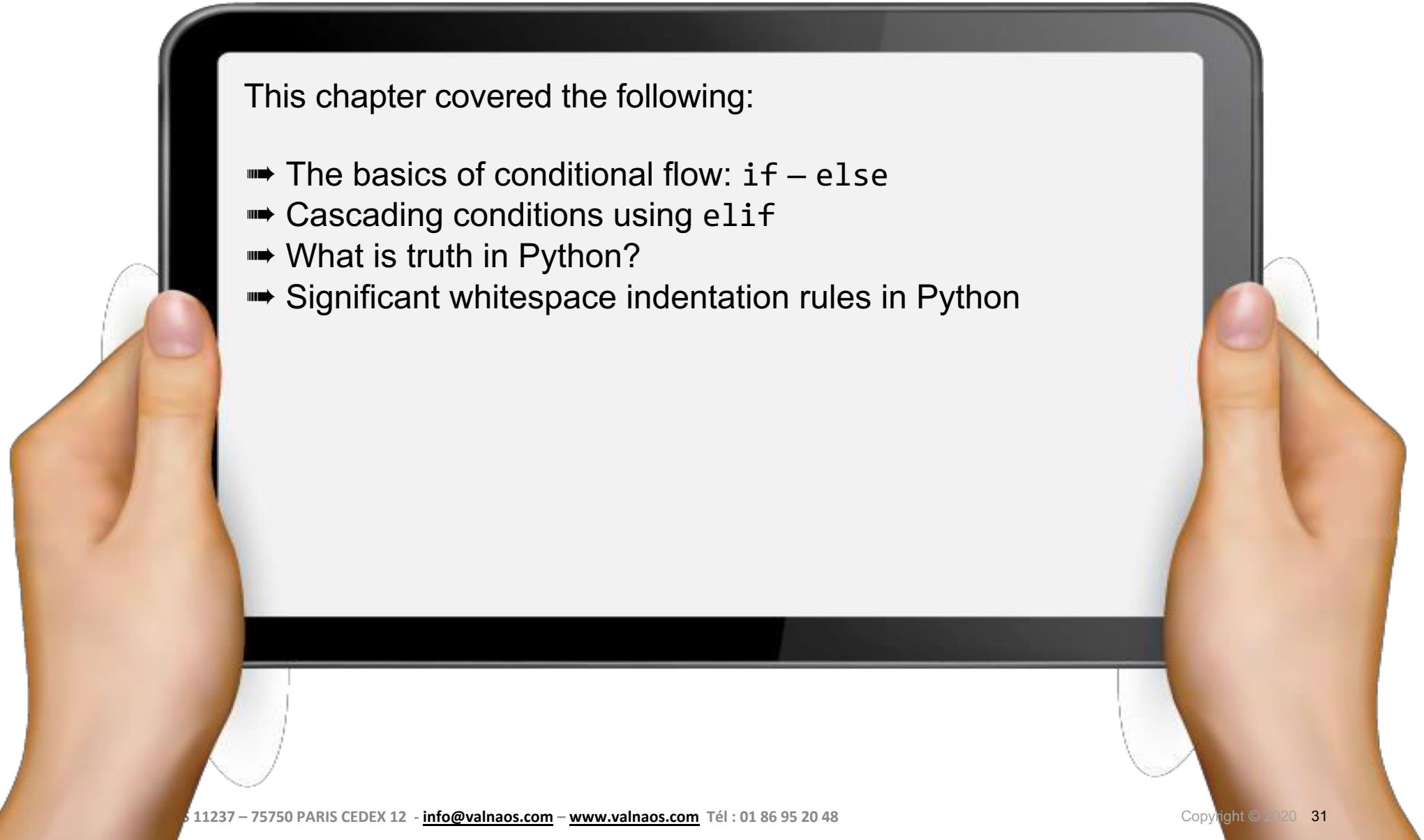
```
12 is even
12/2 = 6
```

Workout Time!



Chapter 2 Exercises link

Chapter Summary



Valnaos
IT Training & Consulting

ALTRAN
.ALTRAN EDUCATION SERVICES

Python 3 Beginner

Chapter **03**

Loops Part I: Iteration

release 1.0.0

A new datatype: `list`

- An ordered list of objects
- Similar to arrays or vectors in other languages
- Written as a list of comma-separated values between square brackets

```
l = [1, 2, 3]
print(l)

l = ['one', 'two', 'three']
print(l)
```

```
[1, 2, 3]
['one', 'two', 'three']
```

A new datatype: `list`

- Like `str`, `list` supports:
  - the `len` function
  - single-item indexing and ranges with `[]`

```
l = ['a', 'list', 'of', 5, 'objects']

print(len(l))
print(l[0])
print(l[:3])
print(l[-2:])
```

```
a
['a', 'list', 'of']
[5, 'objects']
```

A new datatype: `list`

- Like `str`, `list` supports:
  - concatenation with +

```
[1, 2] + [3, 4]
```

```
[1, 2, 3, 4]
```

  - repetition with *

```
5 * ['spam']
```

```
['spam', 'spam', 'spam', 'spam', 'spam']
```

A new datatype: `list`

- Unlike `str`:
  - is *mutable*

```
l = [0, 1, 2]
l[2] = 3
print(l)
```

```
[0, 1, 3]
```

  - can contain objects of different types

```
[True, 'hello', int(6/2)]
```

```
[True, 'hello', 3]
```

Let's write some code!

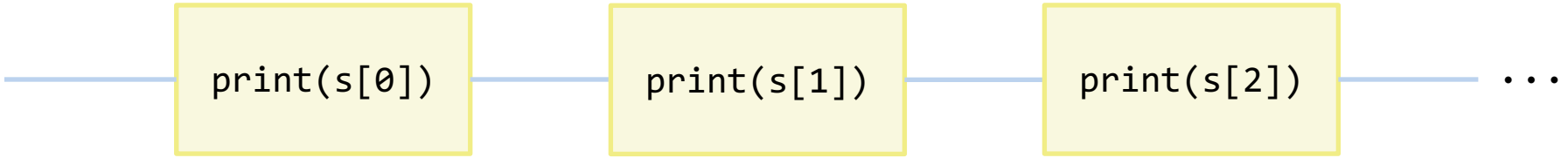
```
l = [1, 2, 3]
print(l)

l = ['one', 'two', 'three']
print(l)
```

```
[1, 2, 3]
['one', 'two', 'three']
```

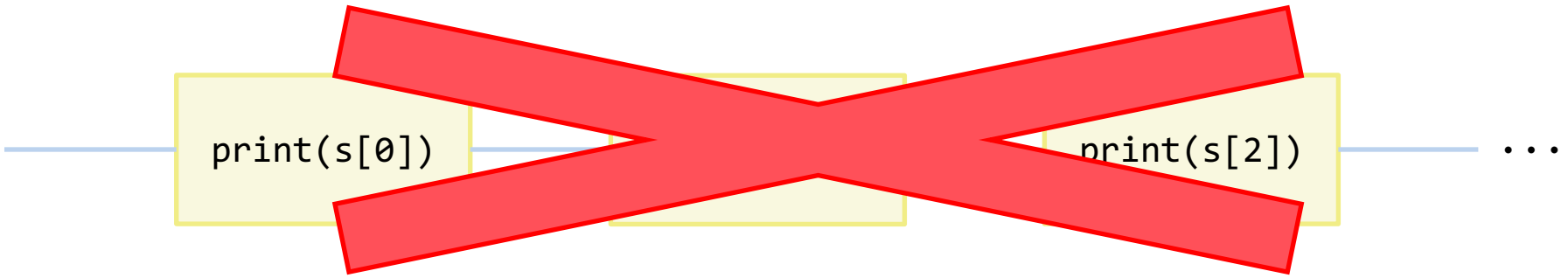
Iteration control flow diagram

➠ How can we print every character in a string?



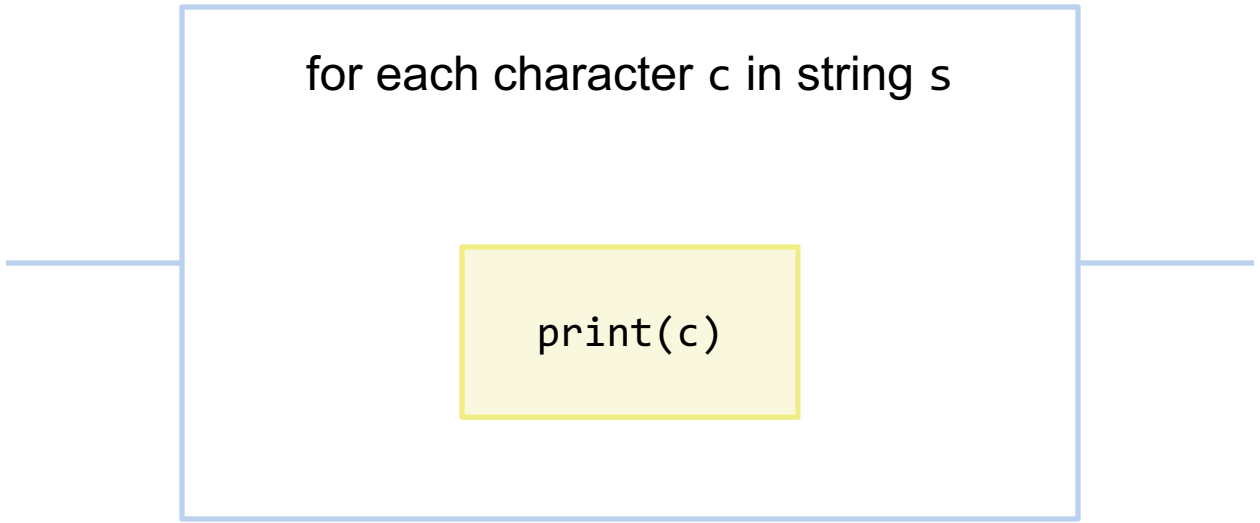
Iteration control flow diagram

➠ How can we print every character in a string?



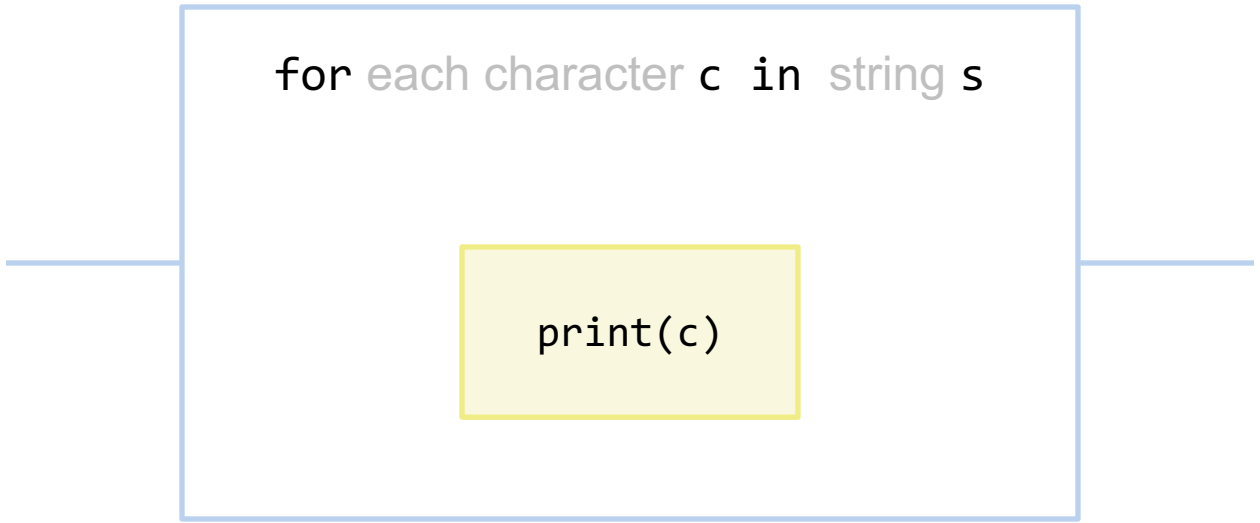
Iteration control flow diagram

➠ How can we print every character in a string?



Iteration control flow diagram

➠ How can we print every character in a string?



Iteration in Python: `for`

➠ `for` *variable* `in` *iterable*`:`

```
s = "Hello World!"

for c in s:
    print(c)
```

```
H
e
l
l
o

W
...
```

Iteration in Python: `for`

➠ `for` *variable* `in` *iterable*`:`

```
s = "Hello World!"

for c in s:
    print(c)
```

```
H
e
l
l
o

W
...
```

Iteration in Python: `for`

➠ `lists` are iterable

```
squares = [0, 1, 4, 9, 16, 25]

for x in squares:
    print(math.sqrt(x))
```

```
0.0
1.0
2.0
3.0
4.0
5.0
```

Iteration in Python: `for`

➠ iterate on a sequence of numbers: `range`

```
for x in range(10):
    print(x)
```

```
0
1
2
3
4
5
6
7
8
9
```

Iteration in Python: for

➠ range is a very versatile function

```
for x in range(10, 15):
    print(x, end=' ')
```

```
10 11 12 13 14
```

```
for x in range(10, 15, 2):
    print(x, end=' ')
```

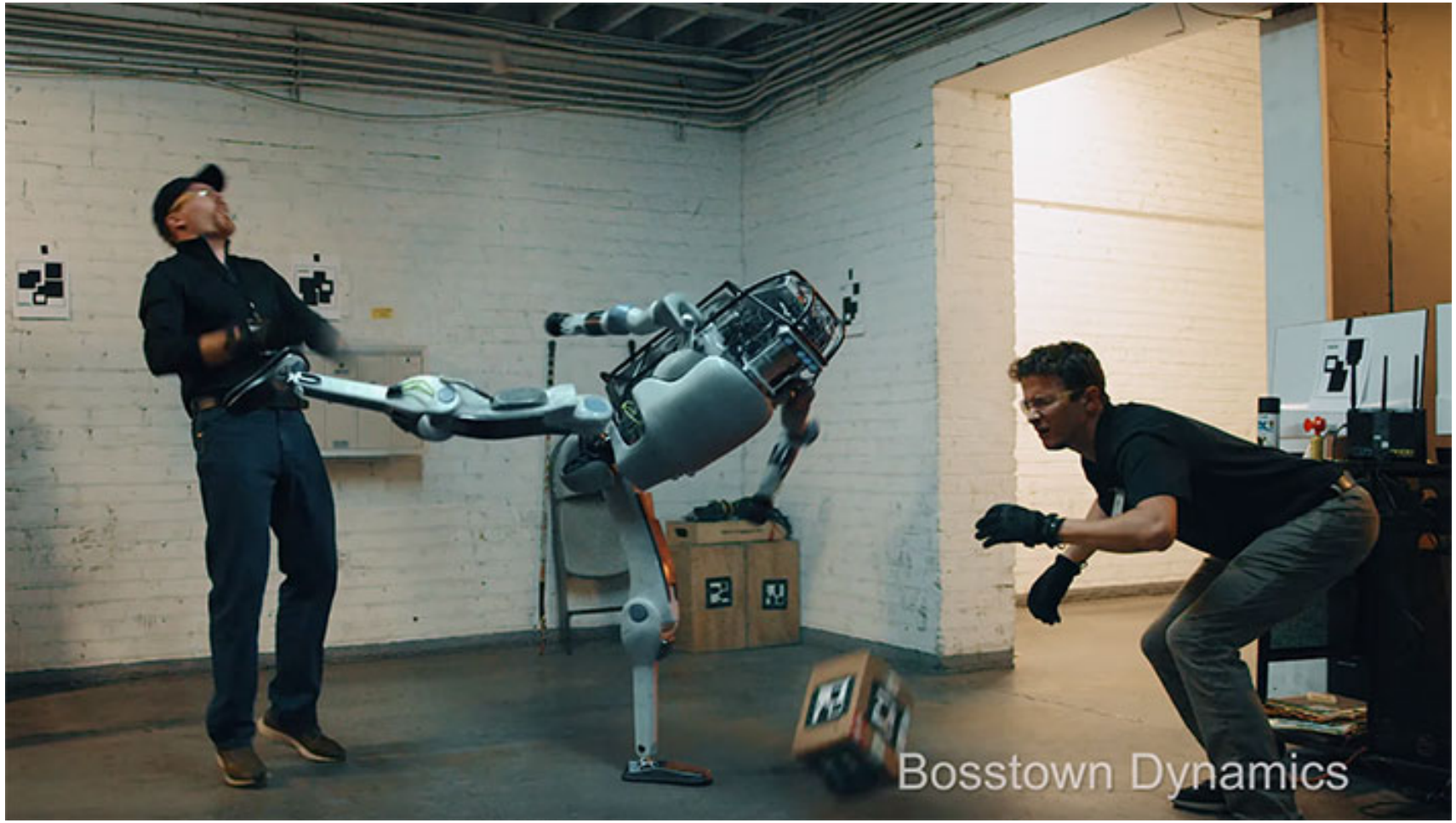
```
10 12 14
```

Let's write some code!

```
for n in [2, 4, 5, 7, 9, 10]:
    if (n % 2) == 0:
        print(f'{n} is even')
    else:
        print(f'{n} is odd')
```

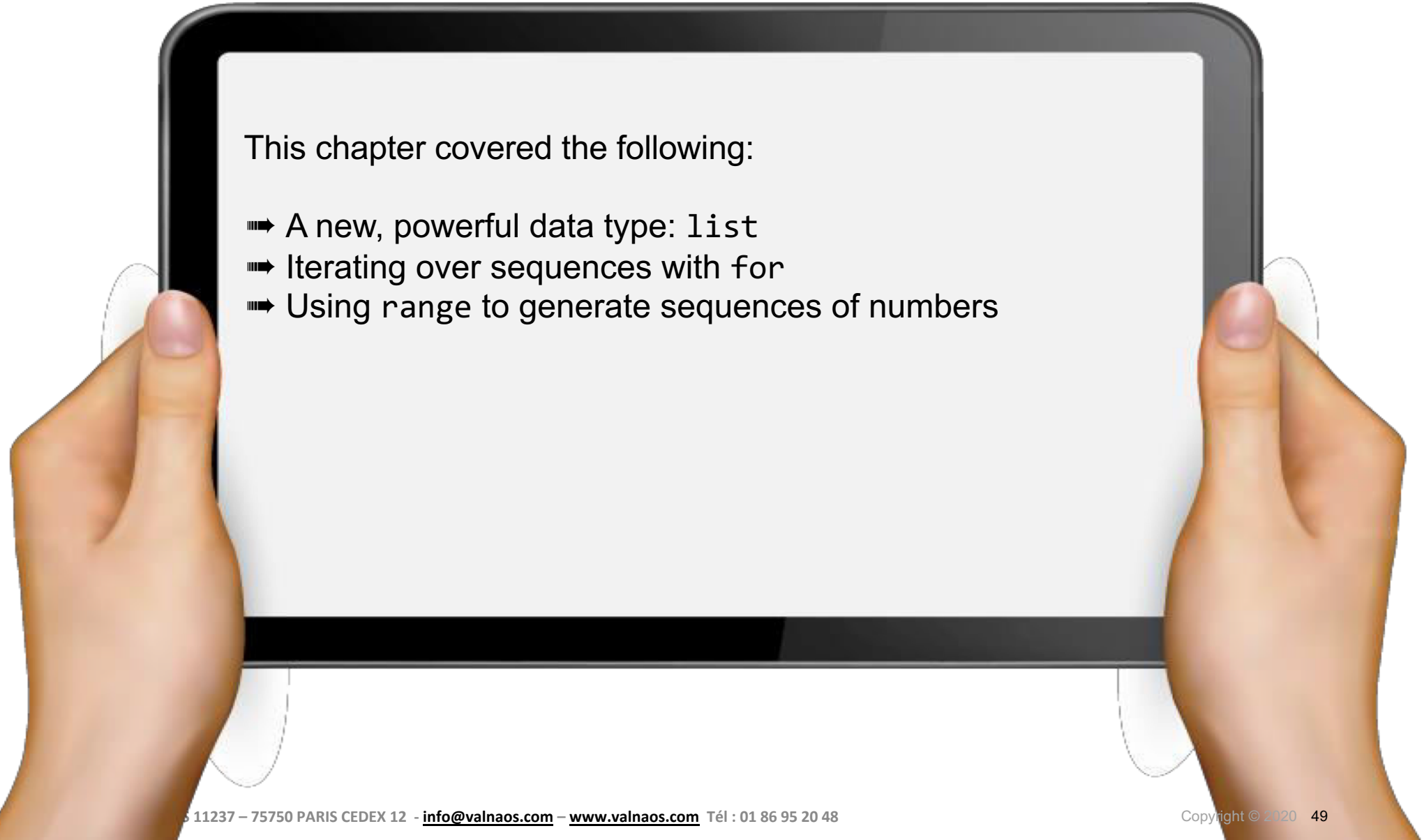
```
2 is even
4 is even
5 is odd
7 is odd
9 is odd
10 is even
```

Workout Time!



Chapter 3 Exercises link

Chapter Summary



Valnaos
IT Training & Consulting

ALTRAN
.ALTRAN EDUCATION SERVICES

Python 3 Beginner

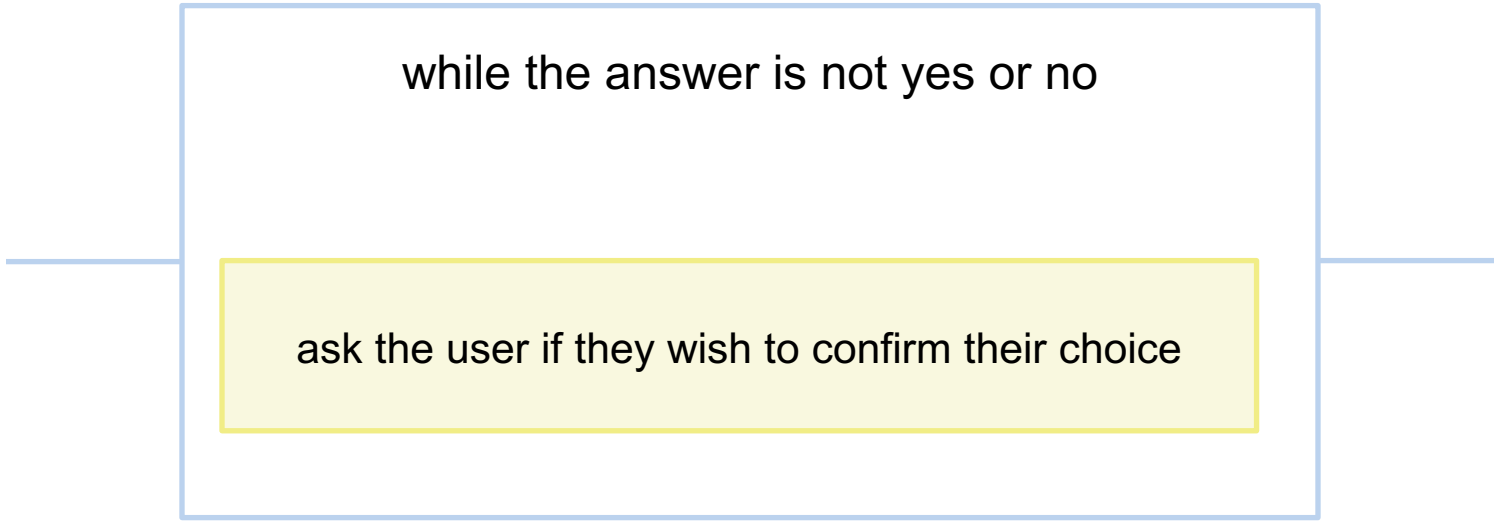
Chapter **04**

Loops Part II: `while`

release 1.0.0

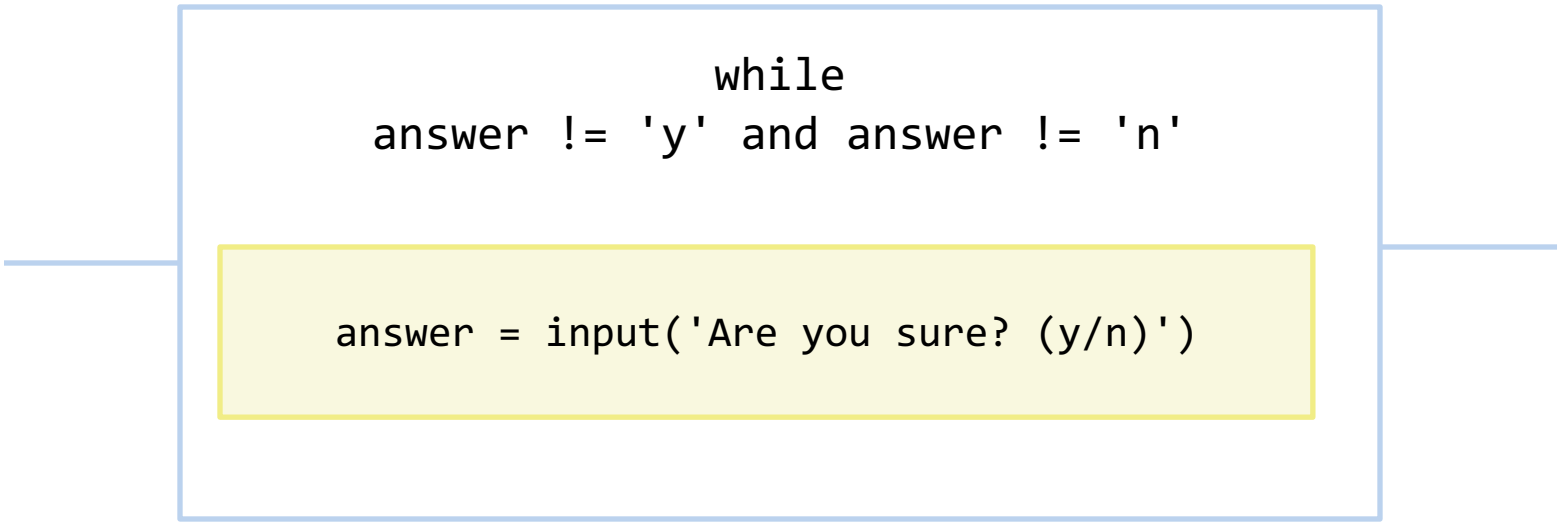
Conditional loop flow diagram

➠ How can we ask a user to confirm a choice?



Conditional loop flow diagram

➠ How can we ask a user to confirm a choice?



Looping in Python: `while`

➠ `while` *`condition`*`:`

```
answer = None

while answer != 'y' and answer != 'n':
    answer = input('Are you sure? (y/n) ')

print(answer)
```

```
Are you sure? (y/n)
Are you sure? (y/n) g
Are you sure? (y/n) yes
Are you sure? (y/n) y
y
```

Looping in Python: `while`

➠ `while` *`condition`*`:`

```
while True:
    print('Hello World!')
```

```
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
...
```

Let's write some code!

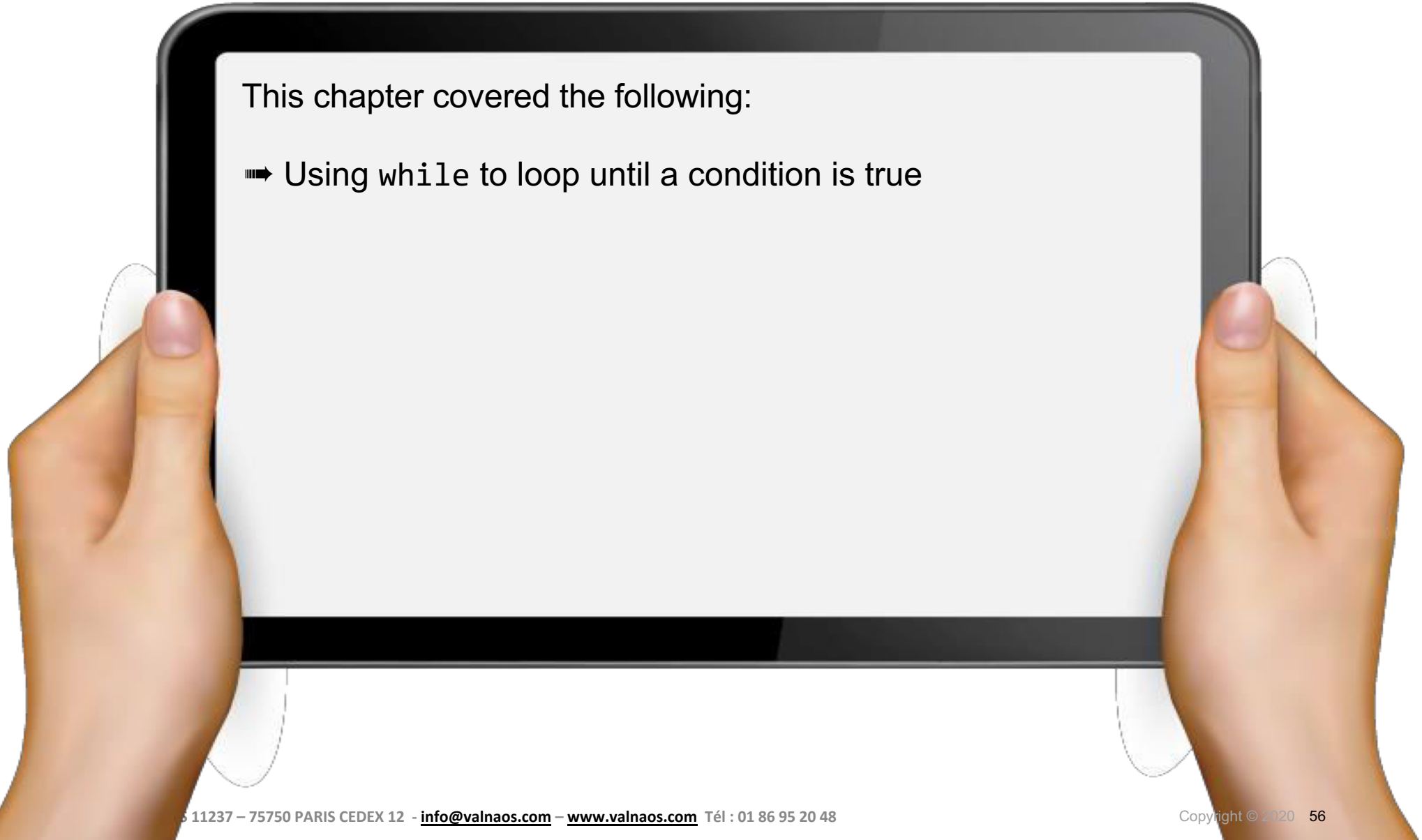
```
answer = None

while answer != 'y' and answer != 'n':
    answer = input('Are you sure? (y/n) ')

print(answer)
```

```
Are you sure? (y/n) y
y
```

Chapter Summary



Python 3 Beginner

Chapter **05**

Mastering control flow

release 1.0.0

"Short-circuiting" a loop

➠ Exit the loop early with `break`

```
for n in [0, 1, 2, -1, 41]:
    if n < 0:
        print(f'invalid value: {n}')
        break
    print(math.sqrt(n))
```

```
0.0
1.0
1.4142135623730951
invalid value: -1
```

"Short-circuiting" a loop

➠ Jump to the next iteration with `continue`

```
addr_list = ['charles@cstar.io', 'tim.apple.com', 'guido@python.org']

for addr in addr_list:
    if '@' not in addr:
        continue
    send_email(addr)        # Sending an email might take a while
    print(f'E-mail sent to {addr}')
```

```
E-mail sent to charles@cstar.io
E-mail sent to guido@python.org
```

"Short-circuiting" a loop

➠ `break` and `continue` also work with `while` loops

```
addr_list = ['charles@cstar.io', 'tim.apple.com', 'guido@python.org']

for addr in addr_list:
    if '@' not in addr:
        continue
    send_email(addr)        # Sending an email might take a while
    print(f'E-mail sent to {addr}')
```

```
E-mail sent to charles@cstar.io
E-mail sent to guido@python.org
```

Loop `else` clause

- Loops can have an `else` clause that is executed if the loop exits without `breaking`

```
for n in [0, 2, 4, 5, 6]:
  if (n % 2) != 0:
    break
  print(f'{n} is even')
else:
  print('All numbers are even')
```

```
0 is even
2 is even
4 is even
```

Iterating over value and index

- Often, we need to iterate over a list while keeping track of the index

```
l = [0, 4, 25]
for i in range(len(l)):
  print(f'Item at position {i} is {l[i]}')
```

```
Item at position 0 is 0
Item at position 1 is 4
Item at position 2 is 25
```

Iterating over value and index

- Often, we need to iterate over a list while keeping track of the index

```
l = [0, 4, 25]
for i in range(len(l)):
  print(f'Item at position {i} is {l[i]}')
```

```
Item at position 0 is 0
Item at position 1 is 4
Item at position 2 is 25
```

Iterating over value and index

➠ Use `enumerate` to update both variables upon looping

```
l = [0, 4, 25]
for i, item in enumerate(l):
  print(f'Item at position {i} is {item}')
```

```
Item at position 0 is 0
Item at position 1 is 4
Item at position 2 is 25
```

Iterating over multiple lists

➠ We can iterate simultaneously over multiple lists using `zip`

```
for i, j in zip(range(5), range(10, 15)):
    print(i, j)
```

```
0 10
1 11
2 12
3 13
4 14
```

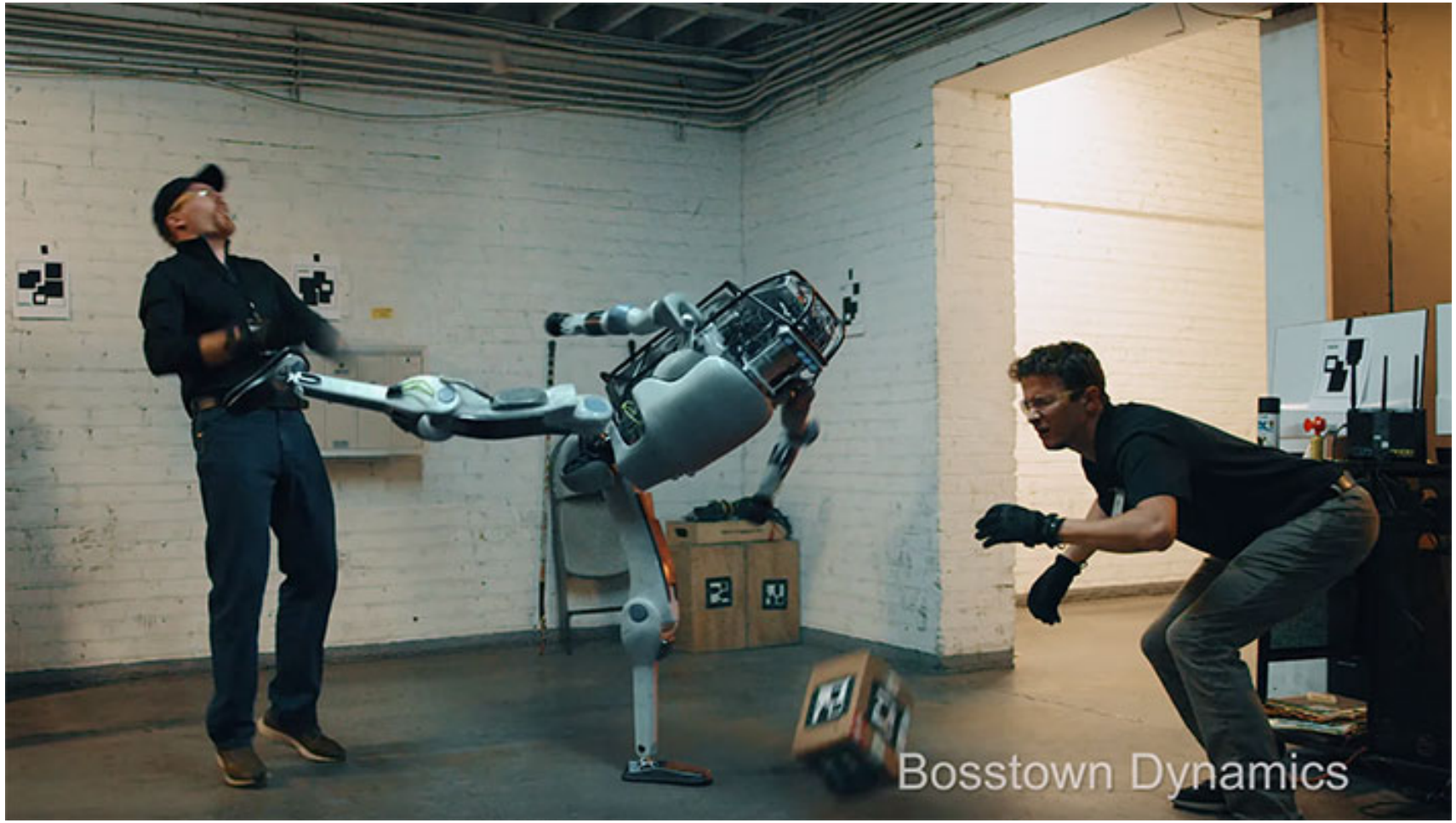
Conditional expressions

- We can return a single value depending on a condition in one simple expression

```
a = 13
b = a / 2 if (a % 2) == 1 else 'odd'
print(b)
```

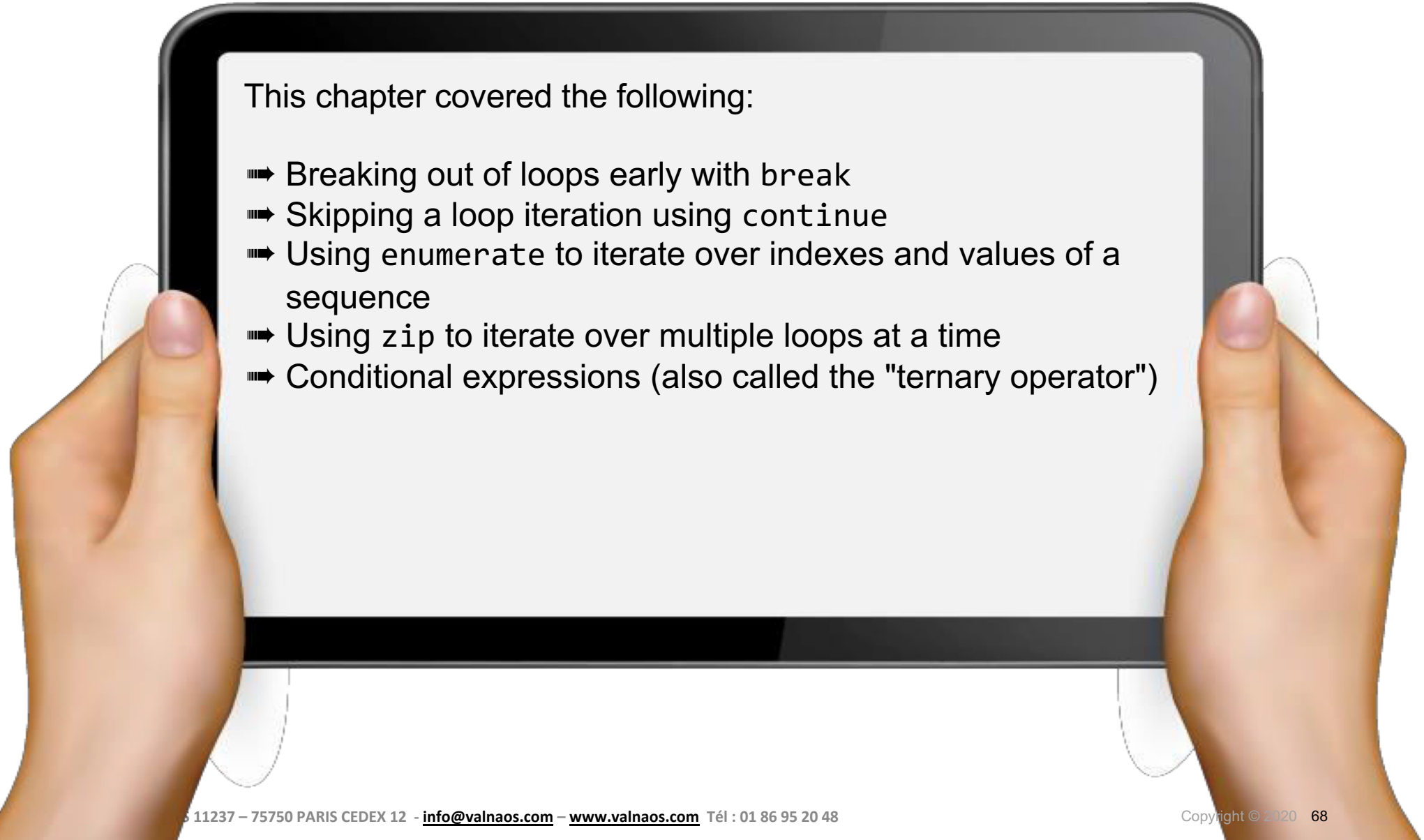
```
odd
```

Workout Time!



Chapter 3 Exercises link

Chapter Summary



Q&A

Workshop

[Session 1 Workshop] link

Thank you for your attention!

