Valnaos
IT Training & Consulting

ALTRAN
.ALTRAN EDUCATION SERVICES

Python 3 Beginner

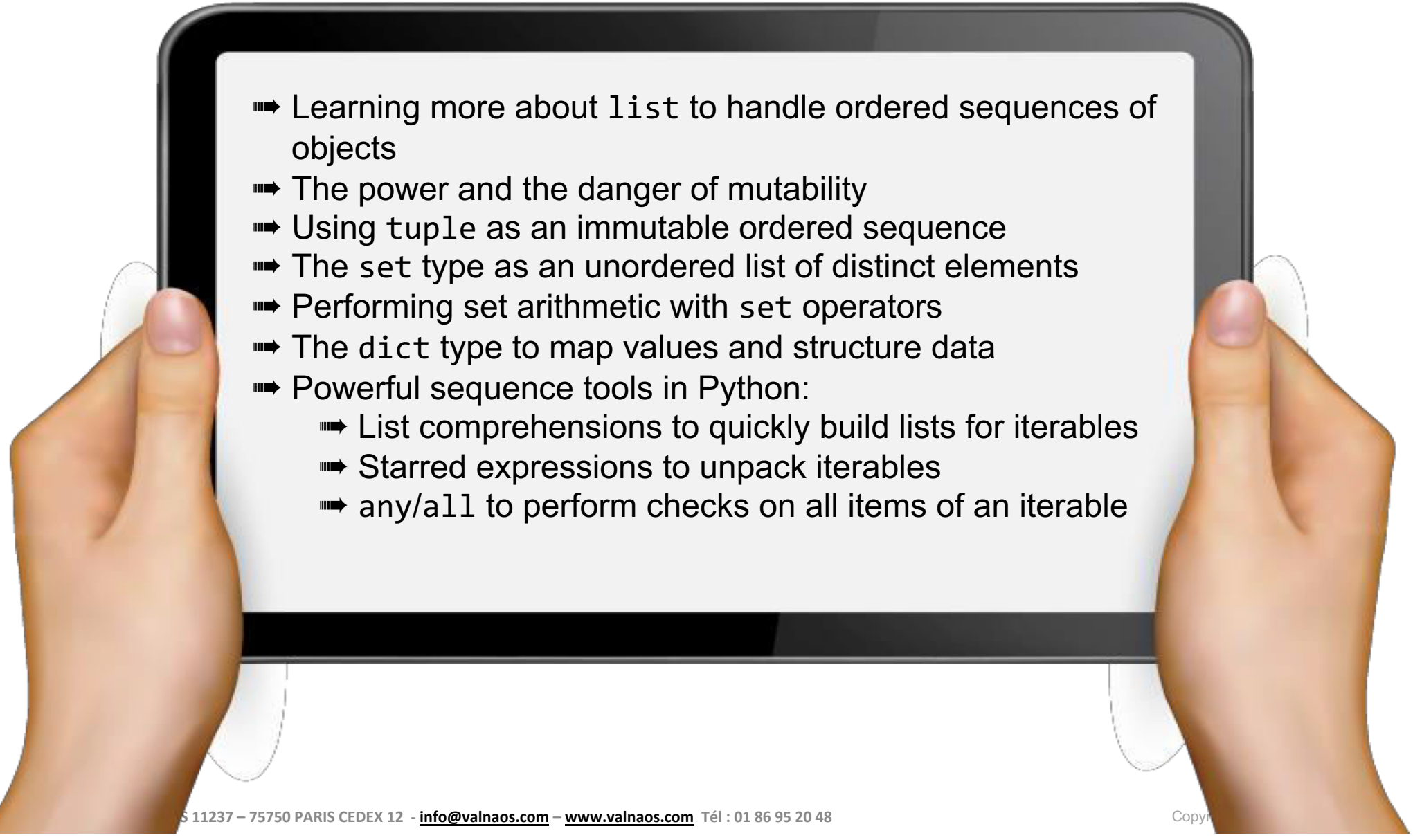
Chapter **00**

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



- Learning more about `list` to handle ordered sequences of objects
- The power and the danger of mutability
- Using `tuple` as an immutable ordered sequence
- The `set` type as an unordered list of distinct elements
- Performing set arithmetic with `set` operators
- The `dict` type to map values and structure data
- Powerful sequence tools in Python:
  - List comprehensions to quickly build lists for iterables
  - Starred expressions to unpack iterables
  - `any`/`all` to perform checks on all items of an iterable

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:

- Define your own functions in Python
- Call a function and use the result
- Know the arcanes of argument passing
- Understand the scope of variables in functions
- Quickly build anonymous functions using lambda expressions
- Understand the basics of functional programming
- Write generator functions to use in your own `for` iterations
- Handle errors in your own code

Session Syllabus

Chapter	Subject	Start	End	Total Time (in hours)
00	Previously on Python 3 Beginner…	14:00	14:15	00:15
01	Functions: Part I	14:15	15:00	00:45
	Coffee Break	*15:00*	*15:10*	*00:10*
02	Functions: Part II	15:10	15:55	00:45
	Coffee Break	*15:55*	*16:05*	*00:10*
03	Generators and Error Handling	16:05	16:50	00:45
	Coffee Break	*16:50*	*17:00*	*00:10*
	Workshop	17:00	18:00	01:00
	Total			**04:00**

But first...

➠ Use `reversed` to iterate a sequence *in reverse*

```
for p in reversed([2, 3, 5, 7, 11, 13]):
    print(p, end=' ')
```

```
13 11 7 5 3 2
```

➠ `sorted` returns a list sorted from the items in an iterable

```
sorted([0, 3, 5, 8, 1, 4, 2, 9, 6, 7])
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
sorted([0, 3, 5, 8, 1, 4, 2, 9, 6, 7], reverse=True)
```

```
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

But first...

➠ `list` has methods to sort or reverse in place

```
l = [1, 6, 3, 9, 7]
l.sort()
print(l)
```

```
[1, 3, 6, 7, 9]
```

```
l.reverse()
print(l)
```

```
[9, 7, 6, 3, 1]
```

Valnaos
IT Training & Consulting

ALTRAN
.ALTRAN EDUCATION SERVICES

Python 3 Beginner

Chapter **01**

Functions: Part I

release 1.0.0

ROT encryption-decryption

```
plaintext = 'Meet me in the park at midnight'

ciphertext = ''
plaintext = plaintext.lower()
for c in plaintext:
    if 'a' <= c <= 'z':
        ciphertext += chr(((ord(c) - ord('a') + 13) % 26) + ord('a'))
    else:
        ciphertext += c

print(ciphertext)
```

ROT encryption-decryption

```
plaintext_0 = 'Meet me in the park at midnight'

ciphertext_0 = ''
plaintext_0 = plaintext_0.lower()
for c in plaintext_0:
    if 'a' <= c <= 'z':
        ciphertext_0 += chr(((ord(c) - ord('a') + 13) % 26) + ord('a'))
    else:
        ciphertext_0 += c

print(ciphertext_0)

plaintext_1 = 'Keep a close eye on Manny'

ciphertext_1 = ''
plaintext_1 = plaintext_1.lower()
for c in plaintext_1:
    if 'a' <= c <= 'z':
        ciphertext_1 += chr(((ord(c) - ord('a') + 5) % 26) + ord('a'))
    else:
        ciphertext_1 += c

print(ciphertext_1)
```

ROT encryption-decryption

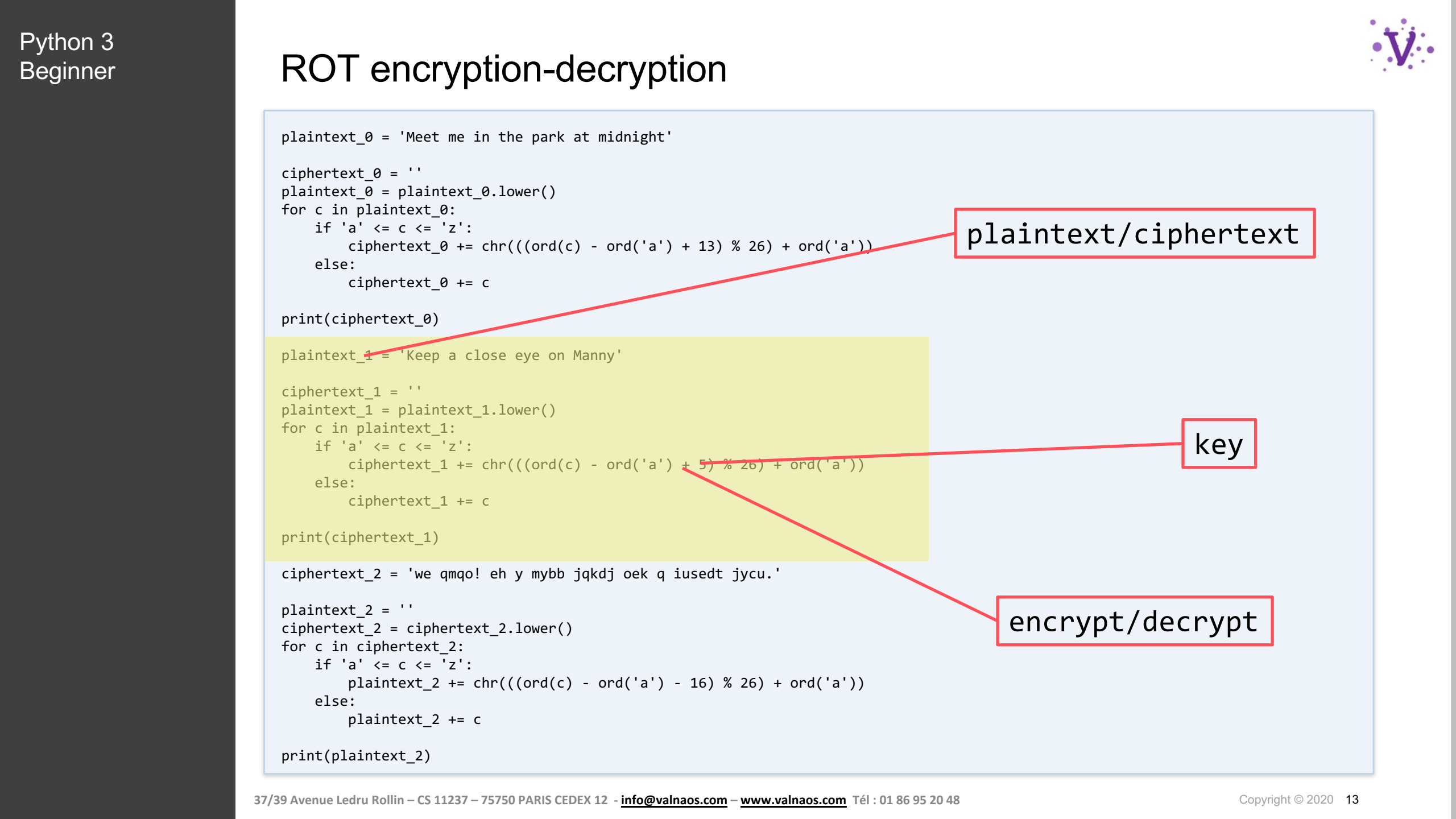
```
plaintext_0 = 'Meet me in the park at midnight'

ciphertext_0 = ''
plaintext_0 = plaintext_0.lower()
for c in plaintext_0:
    if 'a' <= c <= 'z':
        ciphertext_0 += chr(((ord(c) - ord('a') + 13) % 26) + ord('a'))
    else:
        ciphertext_0 += c

print(ciphertext_0)

plaintext_1 = 'Keep a close eye on Manny'

ciphertext_1 = ''
plaintext_1 = plaintext_1.lower()
for c in plaintext_1:
    if 'a' <= c <= 'z':
        ciphertext_1 += chr(((ord(c) - ord('a') + 5) % 26) + ord('a'))
    else:
        ciphertext_1 += c

print(ciphertext_1)

ciphertext_2 = 'we qmqo! eh y mybb jqkdj oek q iusedt jycu.'

plaintext_2 = ''
ciphertext_2 = ciphertext_2.lower()
for c in ciphertext_2:
    if 'a' <= c <= 'z':
        plaintext_2 += chr(((ord(c) - ord('a') - 16) % 26) + ord('a'))
    else:
        plaintext_2 += c

print(plaintext_2)
```

ROT encryption-decryption

```
plaintext_0 = 'Meet me in the park at midnight'

ciphertext_0 = ''
plaintext_0 = plaintext_0.lower()
for c in plaintext_0:
    if 'a' <= c <= 'z':
        ciphertext_0 += chr(((ord(c) - ord('a') + 13) % 26) + ord('a'))
    else:
        ciphertext_0 += c

print(ciphertext_0)
```

```
plaintext_1 = 'Keep a close eye on Manny'

ciphertext_1 = ''
plaintext_1 = plaintext_1.lower()
for c in plaintext_1:
    if 'a' <= c <= 'z':
        ciphertext_1 += chr(((ord(c) - ord('a') + 5) % 26) + ord('a'))
    else:
        ciphertext_1 += c

print(ciphertext_1)
```

```
ciphertext_2 = 'we qmqo! eh y mybb jqkdj oek q iusedt jycu.'

plaintext_2 = ''
ciphertext_2 = ciphertext_2.lower()
for c in ciphertext_2:
    if 'a' <= c <= 'z':
        plaintext_2 += chr(((ord(c) - ord('a') - 16) % 26) + ord('a'))
    else:
        plaintext_2 += c

print(plaintext_2)
```



Defining a function

```
def rot(input_text, key, encrypt):
    if not encrypt:
        key = -key
    output_text = ''
    input_text = input_text.lower()
    for c in input_text:
        if 'a' <= c <= 'z':
            output_text += chr(((ord(c) - ord('a') + key) % 26) + ord('a'))
        else:
            output_text += c
    print(output_text)
```

ROT encryption-decryption

```
def rot(input_text, key, encrypt):
    if not encrypt:
        key = -key
    output_text = ''
    input_text = input_text.lower()
    for c in input_text:
        if 'a' <= c <= 'z':
            output_text += chr(((ord(c) - ord('a') + key) % 26) + ord('a'))
        else:
            output_text += c
    print(ciphertext)

plaintext_0 = 'Meet me in the park at midnight'
rot(plaintext_0, 13, True)

plaintext_1 = 'Keep a close eye on Manny'
rot(plaintext_1, 5, True)

ciphertext_2 = 'we qmqo! eh y mybb jqkdj oek q iusedt jycu.'
rot(ciphertext_2, 16, False)
```

Anatomy of a function declaration

```
def rot(input_text, key, encrypt):
    if not encrypt:
        key = -key
    ...
```

Anatomy of a function declaration

```
def rot(input_text, key, encrypt):
    if not encrypt:
        ...
```

Function declaration keyword

Anatomy of a function declaration

```
def rot(input_text, key, encrypt):
    if not encrypt:
        key = -key
```

Function name

Anatomy of a function declaration

```
def rot(input_text, key, encrypt):
    if not encrypt:
        key = -key
```

Function arguments

Anatomy of a function declaration

```
def rot(input_text, key, encrypt):
    if not encrypt:
        key = -key
    ...
```

Function body

Calling a function

- Call a function using its name, and arguments between parentheses

```
rot(plaintext_0, 13, True)
```

```
zrrg zr va gur cnex ng zvqavtug
```

- Arguments can be any expression
- Arguments can be passed by keyword

```
rot(encrypt=True, input_text=plaintext_0, key=13)
```

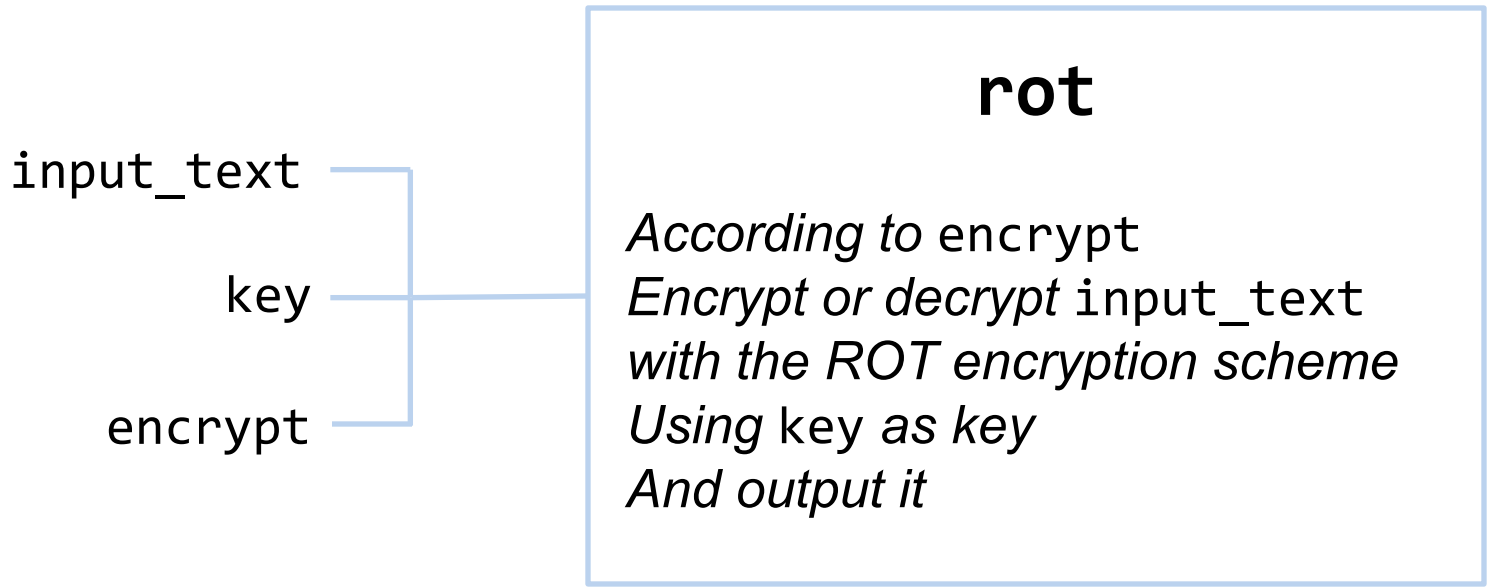
```
zrrg zr va gur cnex ng zvqavtug
```

Let's write some code!

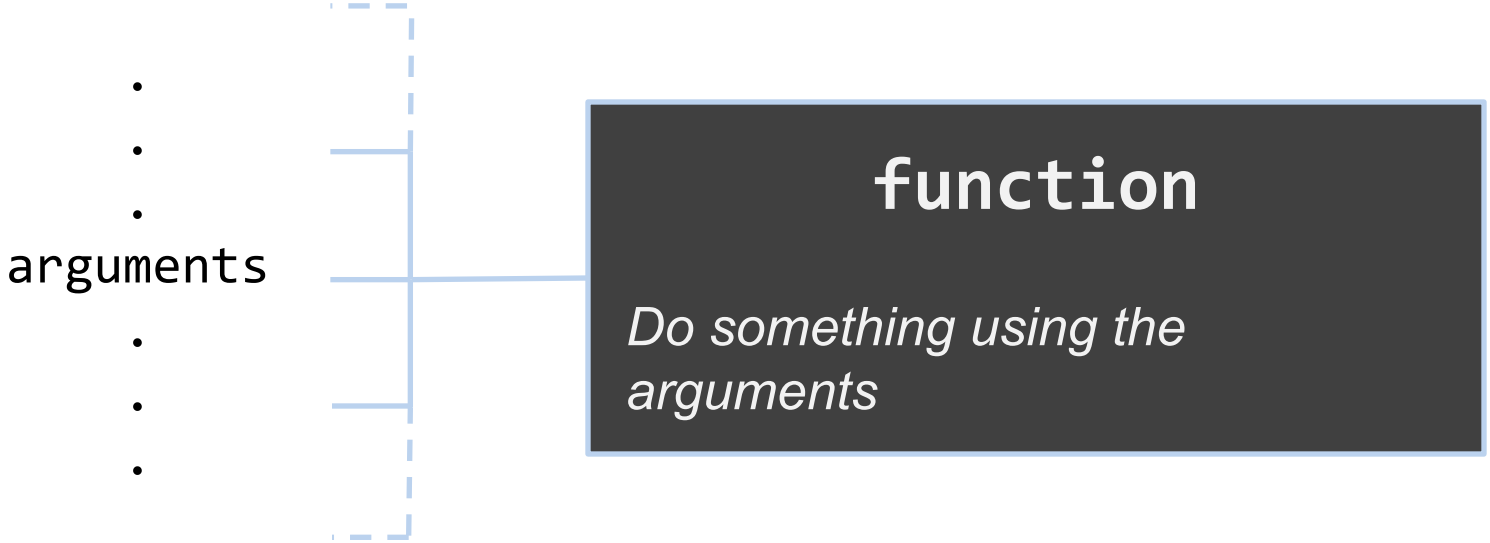
```
def say(text):
    print('I say:', text)

say('Hello')
```

```
I say: Hello
```

Conceptual view of a function



Conceptual view of a function



Functions return a value

- Function calls are expressions that evaluate to a value
- Functions that don't return a value return `None`

```
print(rot(plaintext_0, 13, True))
```

```
zrrg zr va gur cnex ng zvqavtug
None
```

- 👀 Note that our call to `rot()` was used as argument in our call to `print()`

Functions return a value

- Function calls are expressions that evaluate to a value
- Functions that don't return a value return `None`

```
print(rot(plaintext_0, 13, True))
```

```
zrrg zr va gur cnex ng zvqavtug
None
```

- 👀 Note that our call to `rot()` was used as argument in our call to `print()`

Functions return a value

➠ Returning the value instead of printing it

```
def rot(input_text, key, encrypt):
    if not encrypt:
        key = -key
    output_text = ''
    input_text = input_text.lower()
    for c in input_text:
        if 'a' <= c <= 'z':
            output_text += chr(((ord(c) - ord('a') + key) % 26) + ord('a'))
        else:
            output_text += c
    return output_text
```

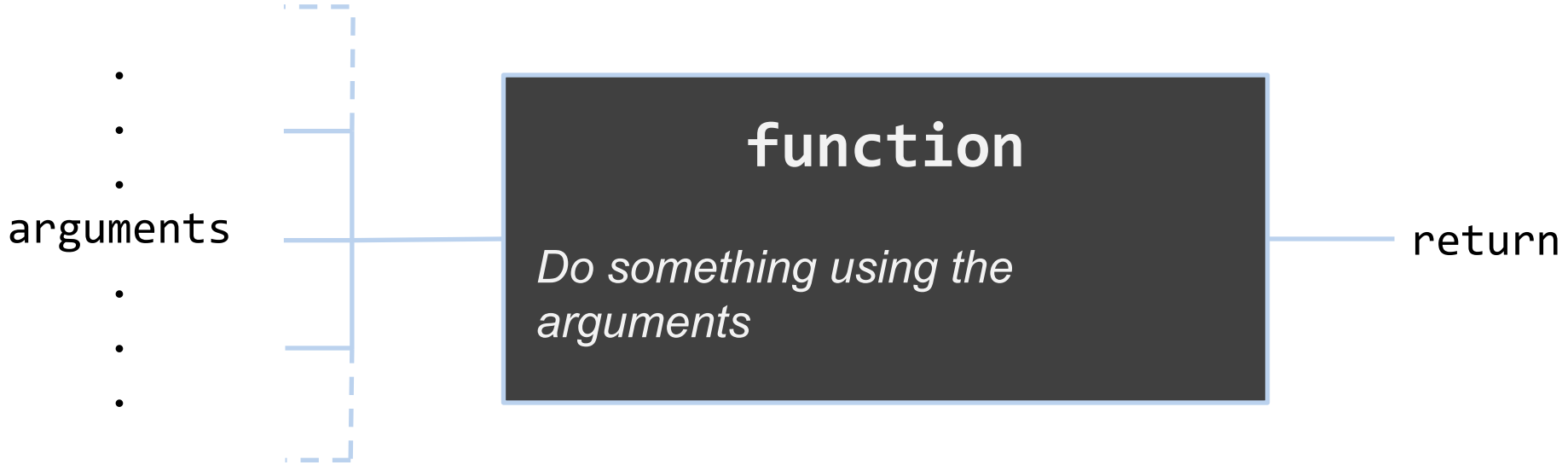
Functions return a value

➠ Assign the return value of a function as a usual expression

```
ciphertext_0 = rot(plaintext_0, 13, True)
```

No output

Conceptual view of a function



Let's write some code!

```
def pretty_format(text, delimiters):
  s = f'{delimiters[0]}{text}{delimiters[1]}'
  return s, len(s)

s, l = pretty_format('Hello World', delimiters=['<== ', ' ==>'])
print(s)
print(f'Length of s: {l}')
```

```
<== Hello World ==>
Length of s: 19
```

Default arguments

➠ Use sensible default values for arguments in function declaration

```
def rot(input_text, key=13, encrypt=True):
    if not encrypt:
        key = -key
    output_text = ''
    input_text = input_text.lower()
    for c in input_text:
        if 'a' <= c <= 'z':
            output_text += chr(((ord(c) - ord('a') + key) % 26) + ord('a'))
        else:
            output_text += c
    return output_text
```

Default arguments

➠ Make function calls easier to read in the most common case

```
rot(plaintext_0)
```

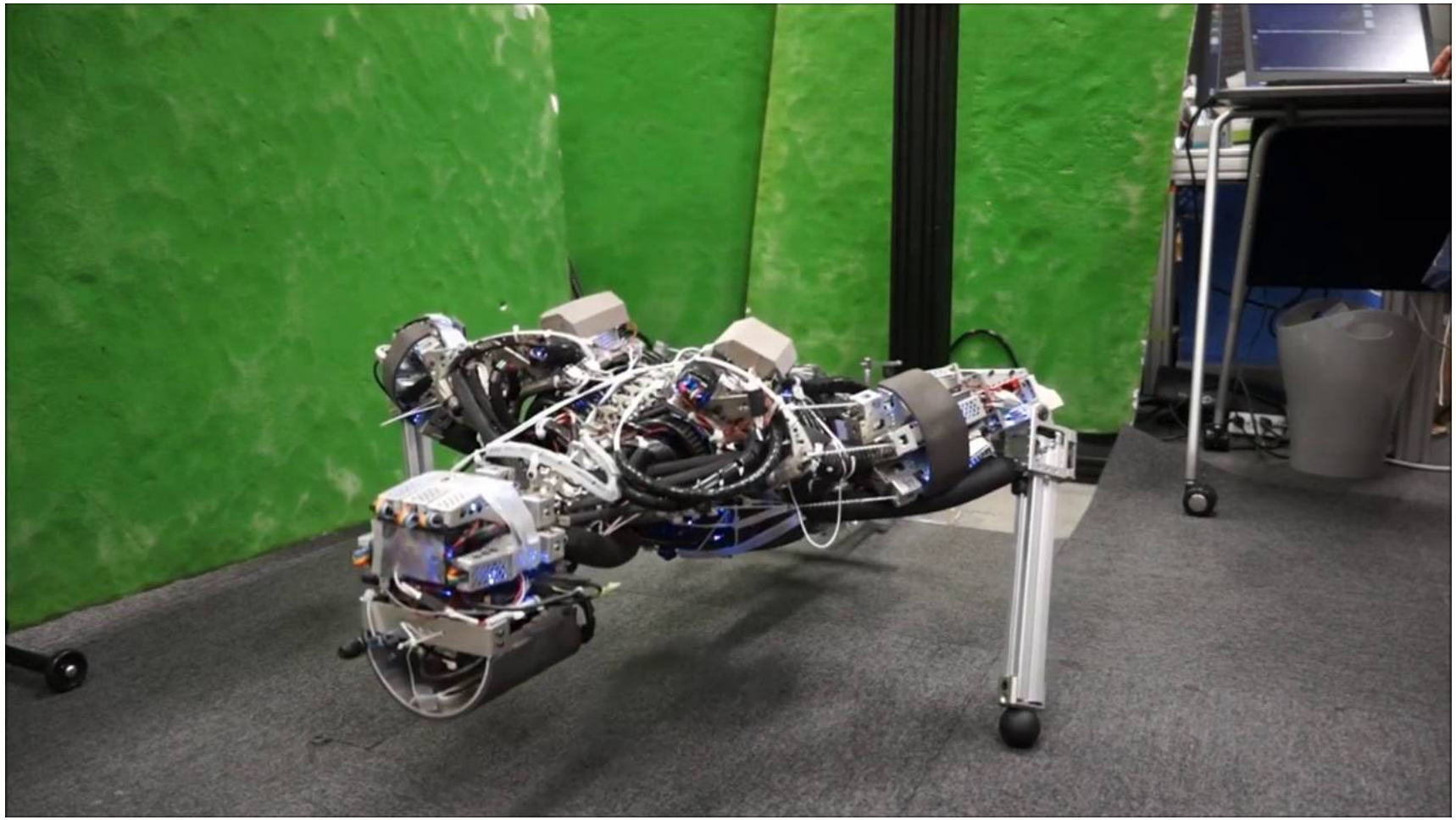
```
'zrrg zr va gur cnex ng zvqavtug'
```

➠ Extend function behaviour without changing every callpoint

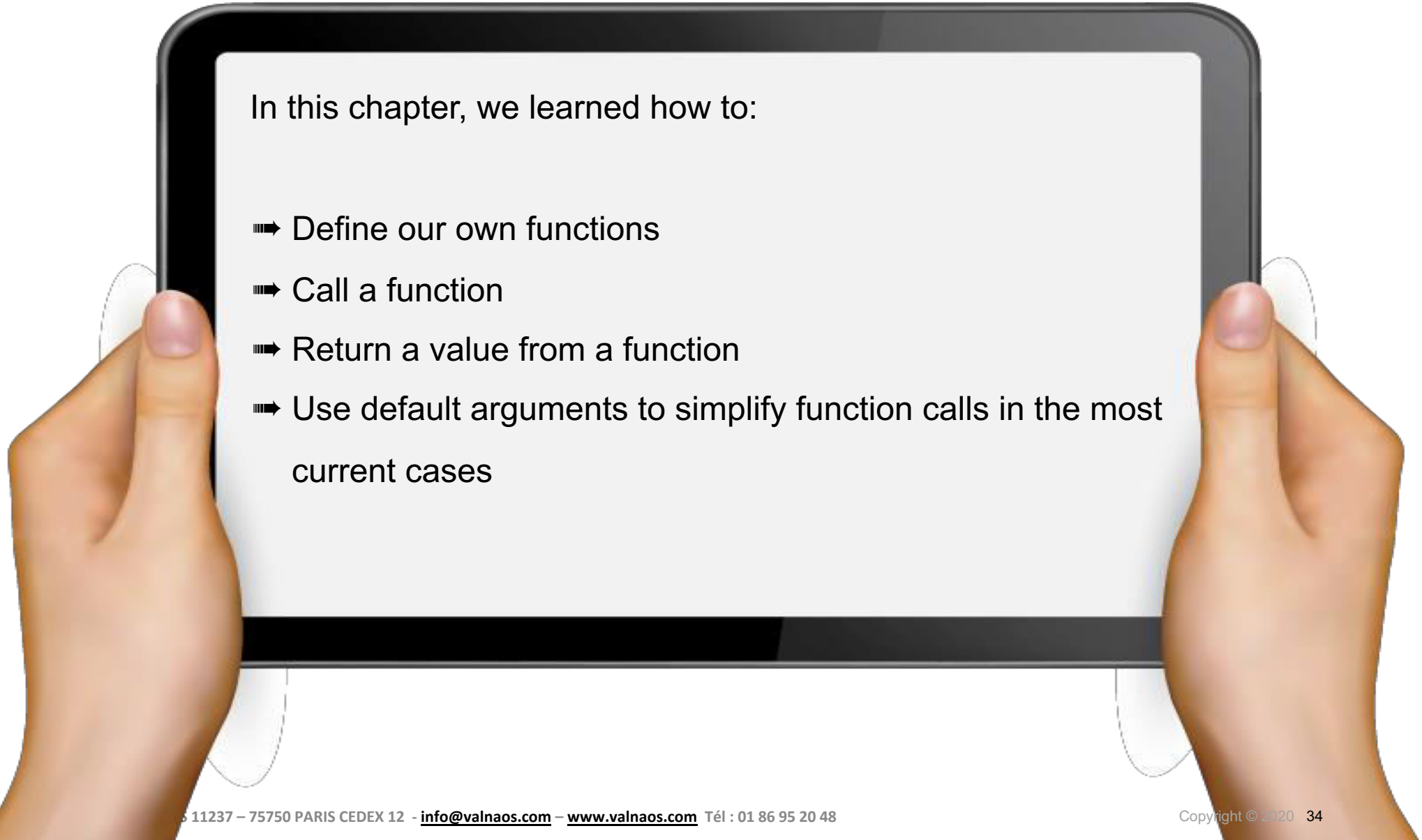
```
def rot(input_text, key=13, encrypt=True, print_it=False):
    ...
    if print_it:
        print(output_text)
    return output_text

ciphertext_0 = rot(plaintext_0, print_it=True)
```

```
zrrg zr va gur cnex ng zvqavtug
```

Workout Time!



Chapter Summary



In this chapter, we learned how to:

- Define our own functions
- Call a function
- Return a value from a function
- Use default arguments to simplify function calls in the most current cases

Valnaos
IT Training & Consulting

ALTRAN
.ALTRAN EDUCATION SERVICES

Python 3 Beginner

Chapter **02**

Functions: Part II

release 1.0.0

Variable arguments

- `print` takes a variable number of arguments

```
print()
print('Hello')
print('Hello', 1, 'World!', [0, 1, 2])
```

- Use `*args` to denote a variable number of arguments
  - `args` is a `list` in your function body
  - Only *positional* arguments (no keywords)

```
def multiple_sqrt(*args):
    return [math.sqrt(n) for n in args]
```

```
multiple_sqrt([0, 1, 4, 9, 16])
[0.0, 1.0, 2.0, 3.0, 4.0]
```

Variable keyword arguments

- Use **kwargs for variable keyword arguments
  - kwargs is a dict in your function body

```
def print_items_with_title(**kwargs):
    for k, v in kwargs.items():
        print(f'{k.title()}: {v}')
```

```
print_items_with_title(greeting='Hello World!', count=12, r=None)
Greeting: Hello World!
Count: 12
R: None
```

Variable arguments

- You can mix `*args` and `**kwargs` in the same function declaration

```
def func(*args, **kwargs):
    # Do stuff with args and kwargs
```

- Variable positional arguments must always come before keyword arguments

```
func(0, 1, 2, hello='world')
```

Unpacking a sequence to arguments

➠ Use * to unpack a `list` as function arguments

```
rot_args = ['Hello World', 1, True]
rot(*rot_args)
```

```
'ifmmp xpsme'
```

➠ Use ** to unpack a `dict` as function keyword arguments

```
rot_args = {'encrypt': False, 'key': 1, 'input_text': 'ifmmp xpsme'}
rot(**rot_args)
```

```
'hello world'
```

Functions and variable scope

➠ Variables created inside a function body only exist inside the function (while it is running)

```
def coin_toss():
    rnd = random()
    if rnd < 0.5:
        return 'Heads'
    else:
        return 'Tails'

coin_toss()
```

```
'Heads'
```

```
print(rnd)
```

```
NameError: name 'rnd' is not defined
```

Functions and variable scope

➠ Arguments are local to function

```
def yell(text):
    text = text.upper()
    print(text + '!')

s = 'hello'
yell(s)
```

```
HELLO!
```

```
print(s)
```

```
hello
```

➠ 👀Note that `s` was not modified by `text = text.upper()`

Functions and variable scope

➠ Variables outside the function can be referenced

```
def yell():
    text = s.upper()
    print(text + '!')

s = 'hello'
yell()
```

```
HELLO!
```

Functions and variable scope

➠ ...But not reassigned

```
def yell():
    s = s.upper()
    print(s + '!')

s = 'hello'
yell()
```

```
UnboundLocalError: local variable 's' referenced before assignment
```

Functions and variable scope

➠ Use global to assign to a variable outside the function

```
def yell():
    global s
    s = s.upper()
    print(s + '!')

s = 'hello'
yell()
```

```
HELLO!
```

➠ ⚠ But watch out for side-effects!

```
print(s)
```

```
HELLO
```

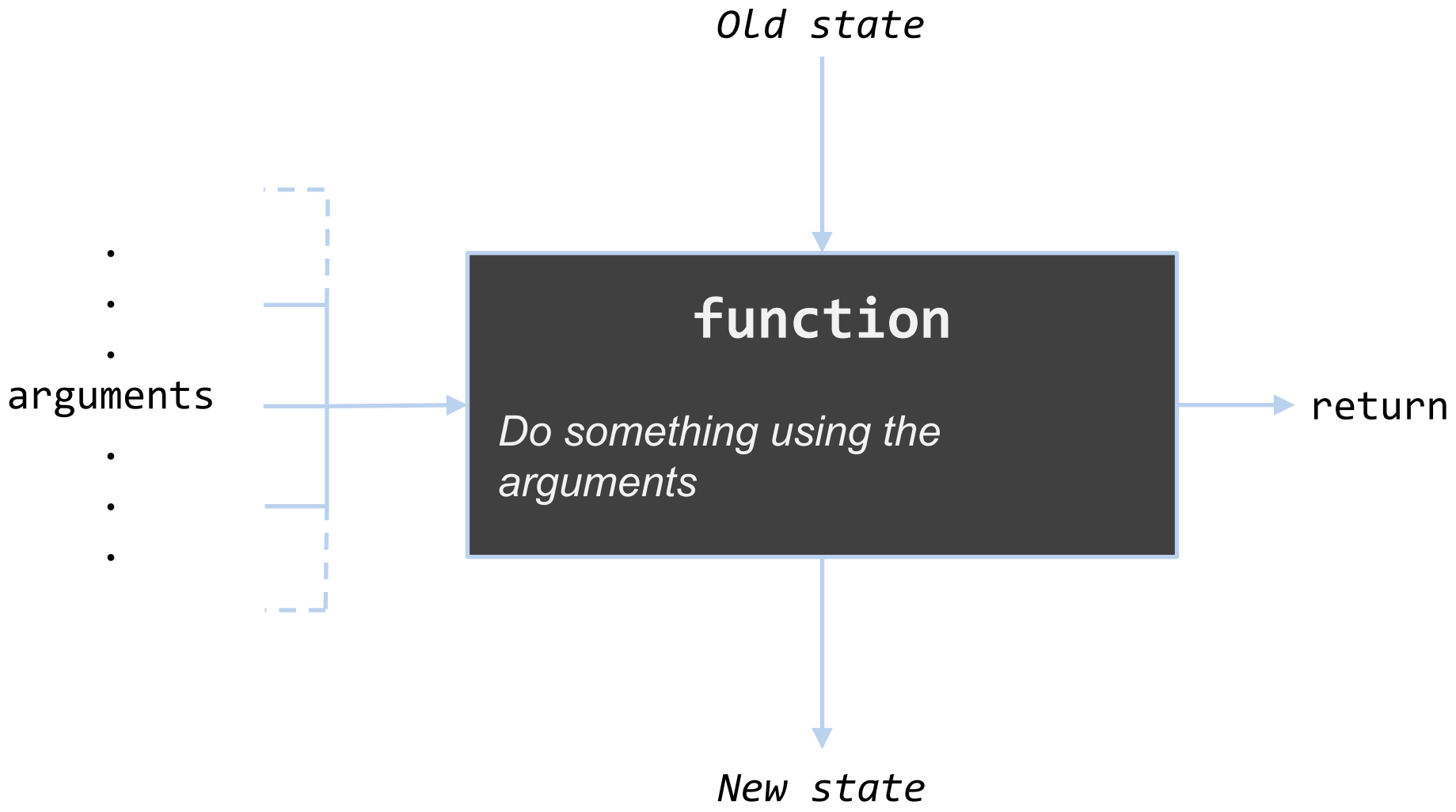
Functions and variable scope

- Variable names are local, but their content is the same object
  - ⚠ Watch out for mutability

```
def add_a_fib(fibs):
    fibs.append(fibs[-1] + fibs[-2])
    print(fibs[-1])
    return fibs

f = [0, 1, 1, 2]
print(add_a_fib(f))

print(f)
```

```
3
[0, 1, 1, 2, 3]

[0, 1, 1, 2, 3]
```

Conceptual view of a function



Let's write some code!

```
def f():
    a = []
    print(a)
    a += [0, 1, 2]
    print(a)

f()
print(a)
```

```
[]
[0, 1, 2]

NameError: name 'a' is not defined
```

Functional programming primer

➠ Functions defined with `def` are just like variables

```
def say_hello():
    print('Hello World!')

type(say_hello)
```

```
function
```

➠ A function can be assigned to another variable

```
f = say_hello
f()
```

```
Hello World!
```

Functional programming primer

➠ A function can be deleted

```
del say_hello

say_hello()
```

```
NameError: name 'say_hello' is not defined
```

➠ A function can be added to a container

```
d = {'obj': 12, 'fun': f}
print(d)
```

```
{'obj': 12, 'fun': <function say_hello at 0x11016bd40>}
```

Functional programming primer

➠ A function can be passed as an argument

```
def do_n_times(fn, n):
    for _ in range(n):
        fn()

do_n_times(f, 3)
```

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

Functional programming primer

- Lambda expressions create anonymous functions from an expression

```
def do_range(fn, n):
    for i in range(n):
        print(fn(i))

do_range(lambda x: x*x, 5)
```

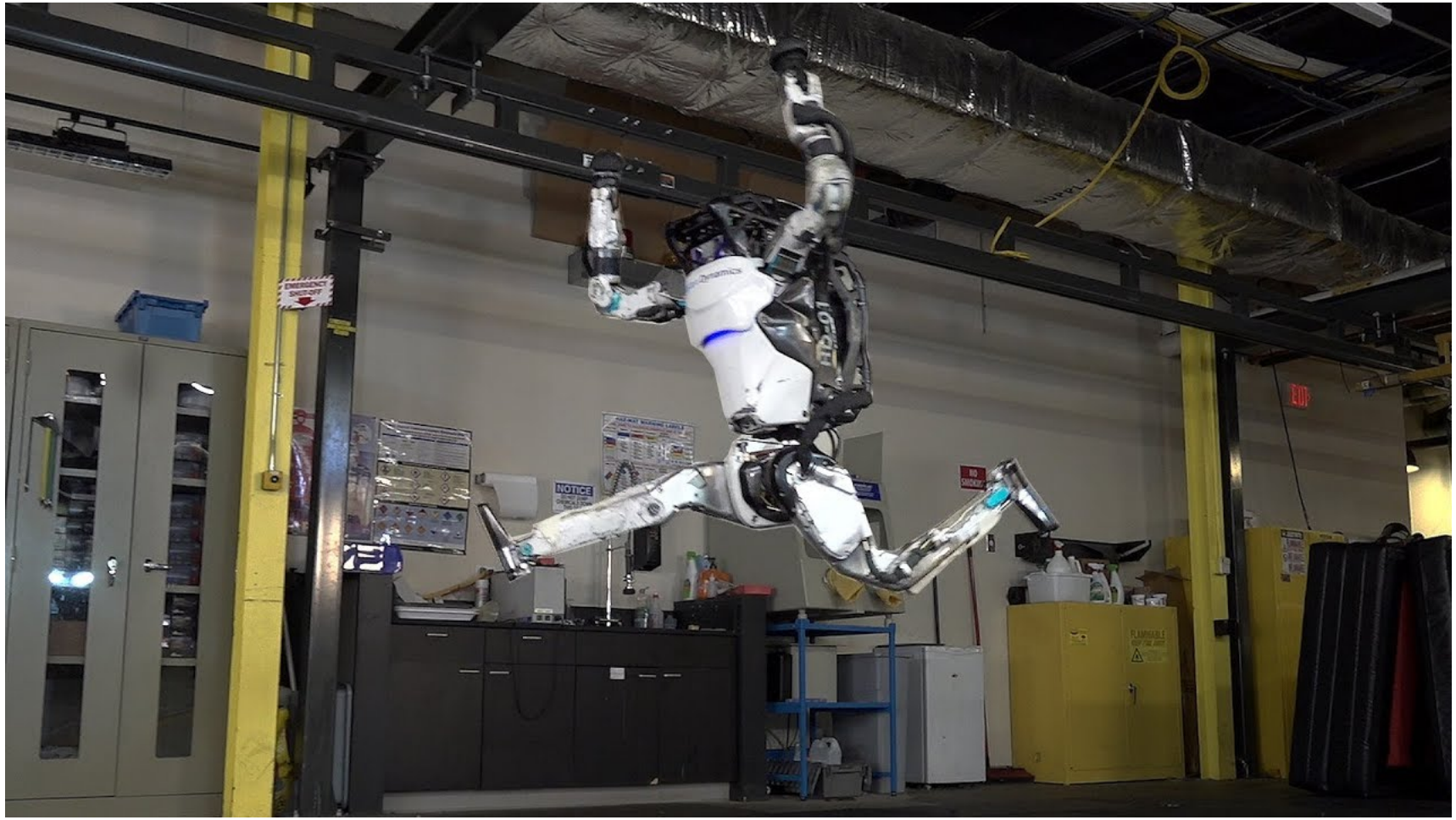
```
0
1
4
9
16
```

Let's write some code!

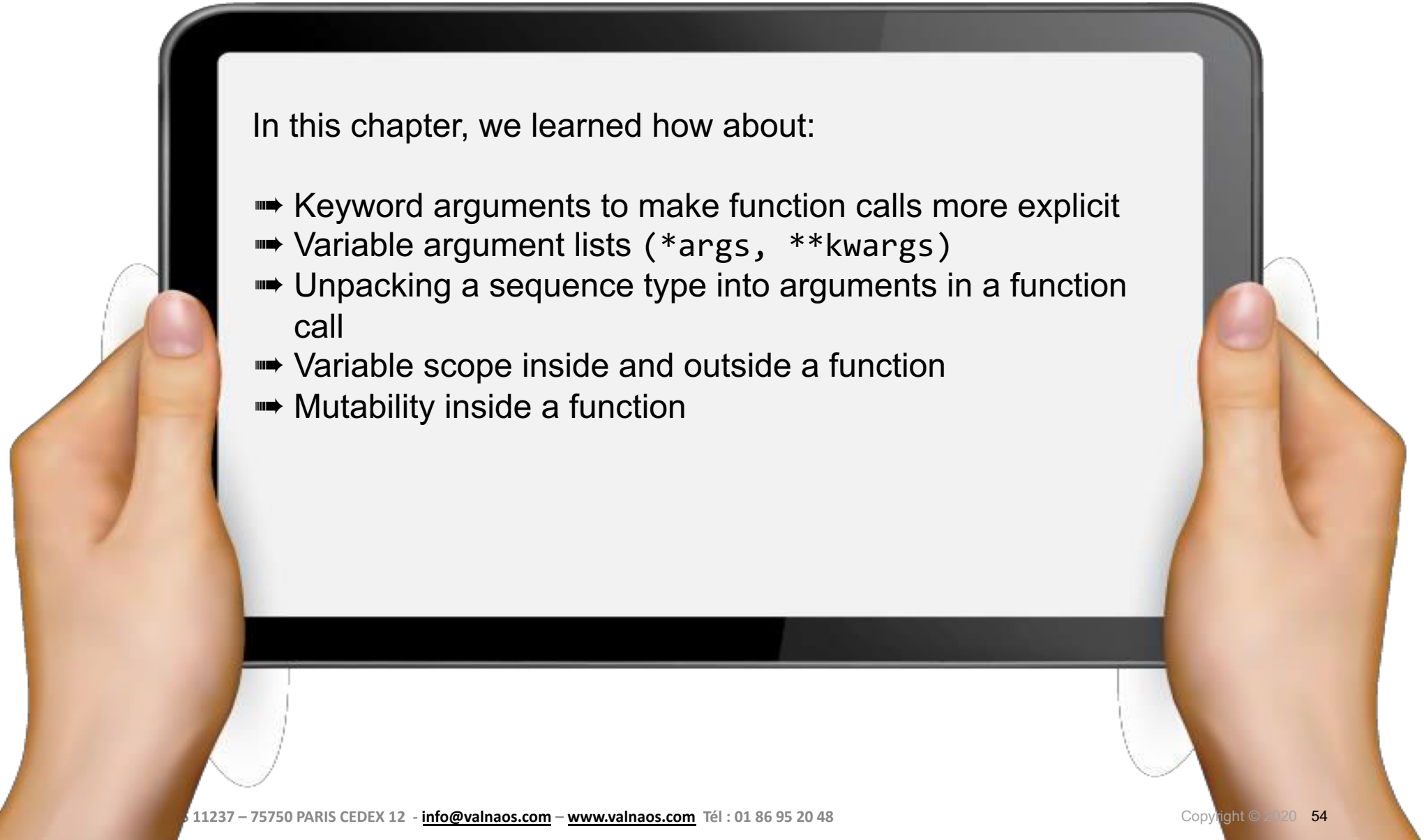
```
def f():
    a = []
    print(a)
    a += [0, 1, 2]
    print(a)

f()
print(a)
```

```
[]
[0, 1, 2]

NameError: name 'a' is not defined
```

Workout Time!



Chapter Summary



In this chapter, we learned how about:

- Keyword arguments to make function calls more explicit
- Variable argument lists (`*args, **kwargs`)
- Unpacking a sequence type into arguments in a function call
- Variable scope inside and outside a function
- Mutability inside a function

Valnaos
IT Training & Consulting

ALTRAN
.ALTRAN EDUCATION SERVICES

Python 3 Beginner

Chapter **03**

Generators and Error handling

release 1.0.0

Generator functions

➠ You can build your own functions to use in a `for` iteration

```
def fib_range(n):
    if n > 0:
        yield 0
    if n > 1:
        yield 1
    f0 = 0
    f1 = 1
    for _ in range(n - 2):
        f = f0 + f1
        yield f
        f0 = f1
        f1 = f
```

Generator functions

➠ Each time the iteration loops, the function runs until the next `yield` statement

```
for f in fib_range(8):
    print(f)
```

```
0
1
1
2
3
5
8
13
```

Iterators

- You can also progress on an iterator using the built-in `next()` function

```
z = zip(firstname, lastnames, dobs)
first, last, dob = next(z)
while first != 'Guido' and last != 'van Rossum':
    first, last, dob = next(z)
    print(f'{first} {last} was born on {dob}')
print('We found Guido!')
```

```
Dennis Ritchie was born on September 9, 1941
Ken Thompson was born on February 4, 1943
Niklaus Wirth was born on February 15, 1934
We found Guido!
```

Errors

➠ You've already seen some errors

```
print(l)
```

```
NameError: name 'l' is not defined
```

```
l = (0, 1, 2)
l.pop()
```

```
AttributeError: 'tuple' object has no attribute 'pop'
```

```
l[0] = 'zero'
```

```
TypeError: 'tuple' object does not support item assignment
```

Errors

➠ Errors "bubble up" function calls

```
def print_sqrt(n):
    print(f'The square root of {n} is {math.sqrt(n)}')

def print_rnd_sqrt():
    print_sqrt(random() - 0.5)

print_rnd_sqrt()
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in print_rnd_sqrt
  File "<stdin>", line 2, in print_sqrt
ValueError: math domain error
```

Catching Errors

➠ Errors "bubble up" function calls

```
def print_sqrt(n):
    try:
        print(f'The square root of {n} is {math.sqrt(n)}')
    except:
        print(f'Could not compute the square root of {n}')

print_rnd_sqrt()
```

```
Could not compute the square root of -0.3587785826661578
```

➠ Open a `try` block to capture errors in unsafe code

➠ Handle errors in an `except` block

Ignoring Errors

➠ Use the `pass` statement to ignore errors in `except` handlers

```
def list_sqrt(in_list):
    out_list = []
    for n in in_list:
        try:
            out_list.append(math.sqrt(n))
        except:
            pass
    return out_list

list_sqrt([2, -1, 16, 'zero', None, 256])
```

```
[1.4142135623730951, 4.0, 16.0]
```

➠ The `pass` statement can be used anywhere, when a block should do nothing

Errors should not be (silently) ignored!

- Use the `pass` statement to ignore errors in `except` handlers

```
def list_sqrt(in_list):
    out_list = []
    for n in in_list:
        try:
            out_list.append(math.sqrt(n))
        except:
            pass
    return out_list

list_sqrt([2, -1, 16, 'zero', None, 256])
```

Never do this, actually! 🚨

```
[1.4142135623730951, 4.0, 16.0]
```

- The `pass` statement can be used anywhere, when a block should do nothing

Catching specific errors

➠ Specify the type of error you wish to catch after `except`

```
x = None
while x is None:
  try:
    x = int(input('Enter an integer number: '))
  except ValueError:
    print(f'Could not parse integer.')
```

```
Enter an integer number: fe
Could not parse integer.
Enter an integer number: ^C
Traceback (most recent call last):
  File "<stdin>", line 3, in <module>
KeyboardInterrupt
```

Catching specific errors

➠ Specify the type of error you wish to catch after `except`

```
x = None
while x is None:
  try:
    x = int(input('Enter an integer number: '))
  except ValueError:
    print(f'Could not parse integer.')
```

```
Enter an integer number: fe
Could not parse integer.
Enter an integer number: ^C
Traceback (most recent call last):
  File "<stdin>", line 3, in <module>
KeyboardInterrupt
```

Deeper into error handling

➠ For the following slides we will be trying to read a csv containing stock prices with the following format

	date	close	volume	open	high	low
2	11:34	270.49	4,787,699	264.50	273.88	262.24
3	2018/10/15	259.5900	6189026.0000	259.0600	263.2800	254.5367
4	2018/10/12	258.7800	7189257.0000	261.0000	261.9900	252.0100
5	2018/10/11	252.2300	8128184.0000	257.5300	262.2500	249.0300
6	2018/10/10	256.8800	12781560.0000	264.6100	265.5100	247.7700
7	2018/10/09	262.8000	12037780.0000	255.2500	266.7700	253.3000
8	2018/10/08	250.5600	13371180.0000	264.5200	267.7599	249.0000
9	2018/10/05	261.9500	17900710.0000	274.6500	274.8800	260.0000
10	2018/10/04	281.8300	9638885.0000	293.9500	294.0000	277.6700
11	2018/10/03	294.8000	7982272.0000	303.3300	304.6000	291.5700
12	2018/10/02	301.0200	11699690.0000	313.9500	316.8400	299.1500
13	2018/10/01	310.7000	21714210.0000	305.7700	311.4400	301.0500
14	2018/09/28	264.7700	33597290.0000	270.2600	278.0000	260.5550

Deeper into error handling

➠ This is the program we wrote

```
def read_csv(path):
    rows = []
    f = open(path)
    s = f.read().splitlines()
    for row in s:
        row = row.split(',')
        date = datetime.strptime(row[0], '%Y/%m/%d')
        price = float(row[1])
        rows.append((date, price))
    f.close()
    return rows
```

➠ This program has many possible errors. Let's dig deeper into it!

Deeper into error handling

```
def read_csv(path):
    rows = []
    f = open(path)
    s = f.read().splitlines()
    for row in s:
        row = row.split(',')
        date = datetime.strptime(row[0], '%Y/%m/%d')
        price = float(row[1])
        rows.append((date, price))
    f.close()
    return rows
```

- Possible errors with the file:
  - File may not exist
  - User may not have read rights to the file
  - Hard drive may be unable to read
  - ...

Deeper into error handling

```
def read_csv(path):
    rows = []
    f = open(path)
    s = f.read().splitlines()
    for row in s:
        row = row.split(',')
        date = datetime.strptime(row[0], '%Y/%m/%d')
        price = float(row[1])
        rows.append((date, price))
    f.close()
    return rows
```

- Possible errors reading lines:
  - `row` may not have data at indexes `0` or `1`
  - date may not be correctly formatted
  - price column may not contain a value convertible to `float`

Deeper into error handling

➠ Let's handle those errors

```
def read_csv(path):
    rows = []
    try:
        f = open(path)
        s = f.read().splitlines()
        for row in s:
            row = row.split(',')
            date = datetime.strptime(row[0], '%Y/%m/%d')
            price = float(row[1])
            rows.append((date, price))
        f.close()
    except OSError as err:
        print(f'Failed to open file. {err}')
    except ValueError as err:
        print(f'Invalid data. {err}')
    except:
        raise
    return rows
```

Deeper into error handling

- Expected errors are handled accordingly
- The error is captured to the err variable and used in the error message

```
except OSError as err:
    print(f'Failed to open file. {err}')
```

```
except ValueError as err:
    print(f'Invalid data. {err}')
```

- Unexpected errors are raised to the caller

```
except:
    raise
```

Deeper into error handling

➠ Use `else` to handle the case when no errors are caught

```
def read_csv(path):
    ...
    except OSError as err:
        print(f'Failed to open file. {err}')
    except ValueError as err:
        print(f'Invalid data. {err}')
    except:
        raise
    else:
        return rows
    return []
```

Deeper into error handling

➠ Execute cleanup code regardless of errors with `finally`

```
def read_csv(path):
    ...
    except OSError as err:
        print(f'Failed to open file. {err}')
    except ValueError as err:
        print(f'Invalid data. {err}')
    except:
        raise
    else:
        return rows
    finally:
        f.close()
    return []
```

Raising your own errors

- You can raise your own errors in functions, for better control

```
def do_something_with_an_int(n):
    if type(n) != int:
        raise TypeError(f'Expected int, got {type(n)}')
    if (n % 2) != 0:
        raise ValueError(f'{n} can't be divided by 2')
    ...
```

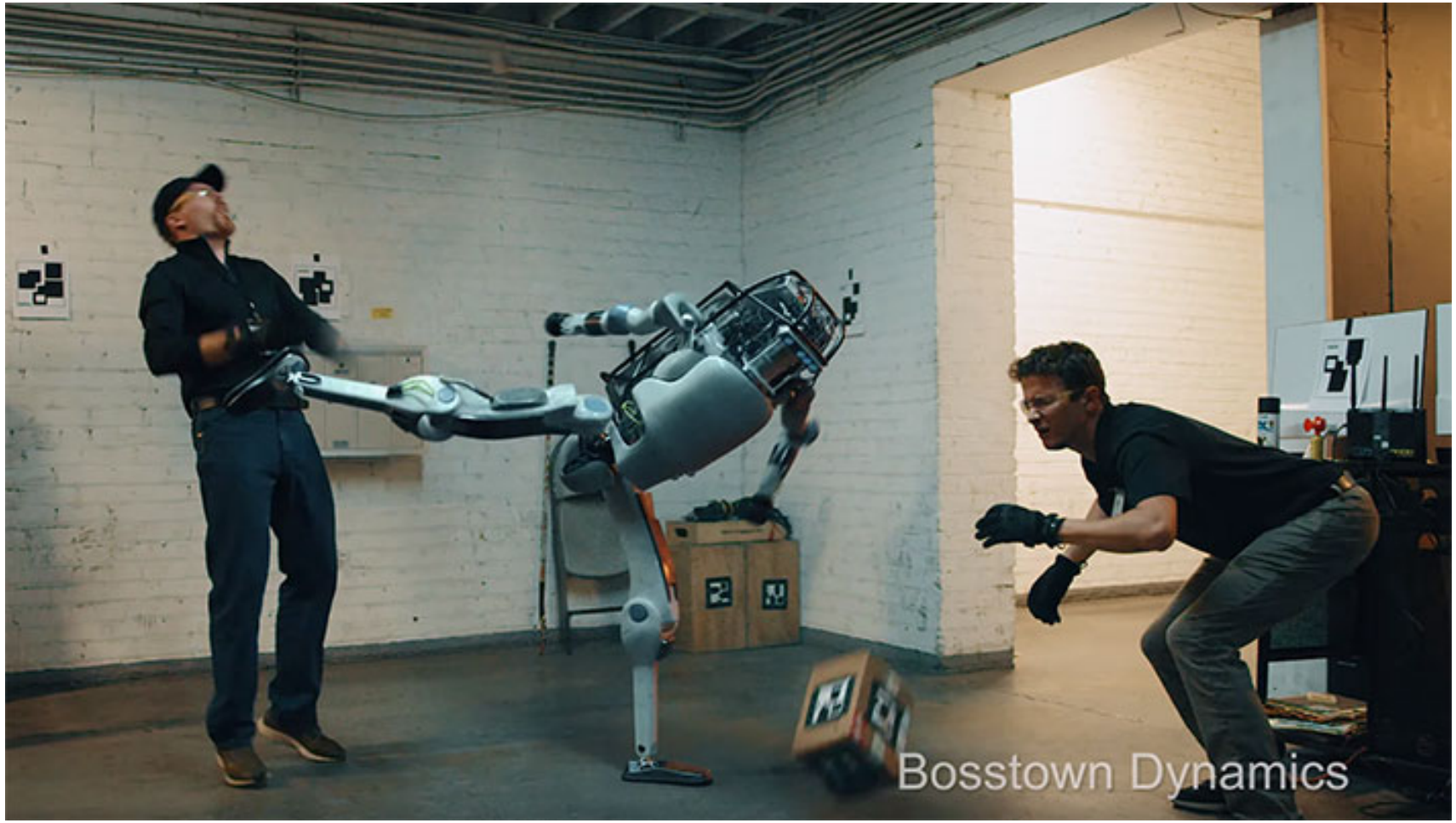
- Learn more about all the built-in Python errors by reading the documentation

Let’s write some code!

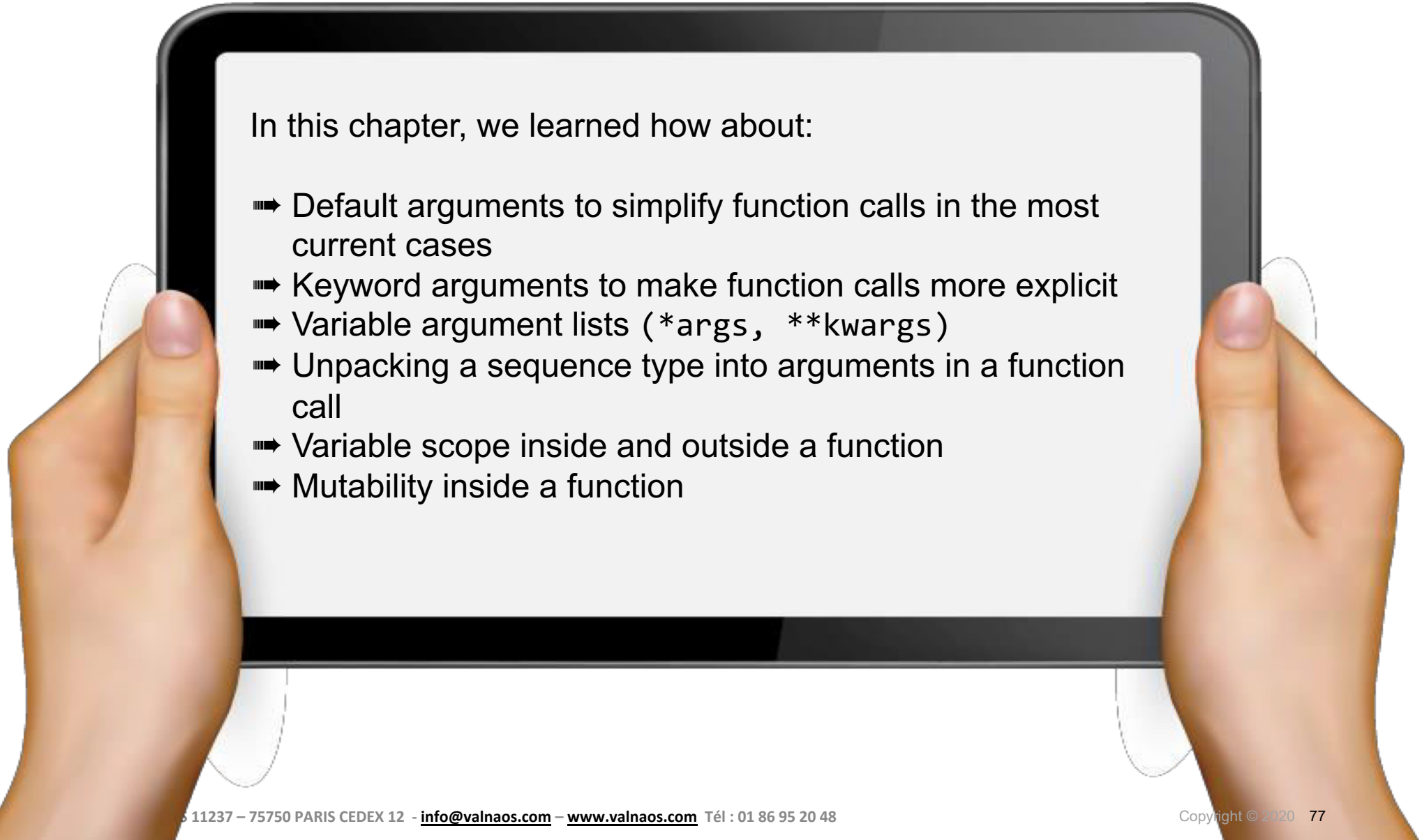
```
s = 'Hello World!'

{c: s.count(c) for c in s}
```

```
{' ': 1, '!': 1, 'H': 1, 'W': 1, 'd': 1, 'e': 1, 'l': 3, 'o': 2, 'r':
1}
```

Workout Time!



Chapter Summary



In this chapter, we learned how about:

- Default arguments to simplify function calls in the most current cases
- Keyword arguments to make function calls more explicit
- Variable argument lists (`*args, **kwargs`)
- Unpacking a sequence type into arguments in a function call
- Variable scope inside and outside a function
- Mutability inside a function

Q&A

Workshop

Thank you for your attention!

