CMU Portugal Academy

CMU Portugal
Advanced Training Program

Foundations of Data Science

DAVID SEMEDO
RAFAEL FERREIRA
NOVA SCHOOL OF SCIENCE AND TECHNOLOGY

Carnegie Mellon Portugal · TÉCNICO LISBOA · Ciências ULisboa · NOVA SCHOOL OF SCIENCE & TECHNOLOGY

Email: df.semedo@fct.unl.pt

David Semedo– Short bio

df.semedo@fct.unl.pt

Assistant Professor @ NOVA FCT, Integrated Researcher @ NOVA LINCS

AI for vision and language. Neural approaches to conversational and contextualized media understanding.

Rafael Ferreira– Short bio

rah.ferreira@campus.fct.unl.pt

4th year PhD Student @ NOVA FCT

Conversational AI. Team leader of the award-winning TWIZ in the Alexa TaskBot Challenge.

CMU Portugal Academy

Today's Topics

1. MODULE STRUCTURE AND LOGISTICS

2. INTRO TO DATA SCIENCE

3. INTRO TO PYTHON

4. DEALING WITH NUMERICAL DATA

CMU Portugal Academy

What will you accomplish today?

- Assess Driver's License Eligibility
- Processing Sensor Data
- Sales performance evaluation
- Costumer churn prediction

CMU Portugal Academy

01

Module Structure and Logistics

Carnegie Mellon Portugal · TÉCNICO LISBOA · Ciências ULisboa · NOVA SCHOOL OF SCIENCE & TECHNOLOGY

CMU Portugal Academy

Module Schedule and Sessions

- 4 Data Science + Python weeks!

- Hands-On Course

	Friday	Saturday
9:00 - 10:00		**Theory + Lab** Lecture + Hands-on + Team-based Work Exercises
10:00 - 11:00		
11:00 - 12:00		
12:00 - 13:00		
13:00 - 16:30		
16:30 - 17:00	**Theory + Lab** Lecture + Hands-on	
17:00 - 18:00		
18:00 - 19:00		
19:00 - 20:00		

CMU Portugal Academy

Module Topics

- Introduction to Data Science and Python Programming
- Statistics and Probability
- Data Preparation and Processing with Pandas
- Machine Learning Fundamentals
- Model Evaluation and Selection
- Data Visualization

CMU Portugal Academy

Goals

- Understand the foundations and applications of Data Science.
- Learn to identify and extract insights from data using Python tools like numpy, pandas and visualization libraries.
- Apply foundational techniques to build simple predictive models.

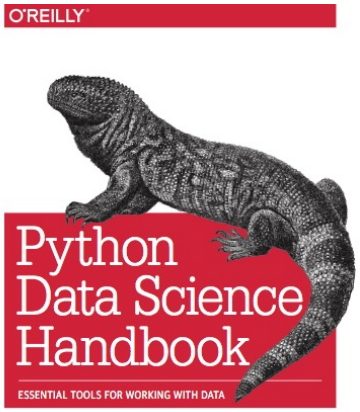
CMU Portugal Academy

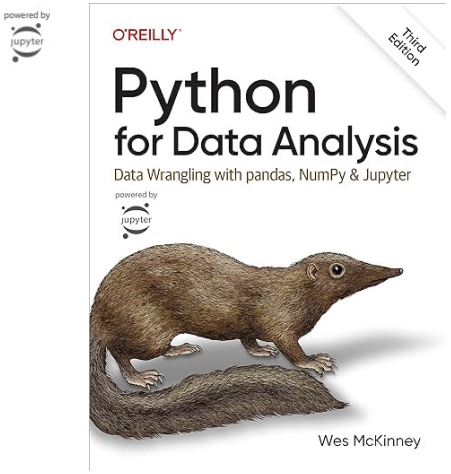
Bibliography and References

- Jake VanderPlas. 2016. **Python Data Science Handbook: Essential Tools for Working with Data** (1st. ed.). O'Reilly Media, Inc.

- Wes McKinney. 2022. **Python for Data Analysis: Data Wrangling with pandas, NumPy, and Jupyter** (3rd. Ed.). O'Reilly Media, Inc.

- Other tools and resources shared throughout the course.





Carnegie Mellon Portugal · TÉCNICO LISBOA · Ciências ULisboa · NOVA SCHOOL OF SCIENCE & TECHNOLOGY

CMU Portugal Academy

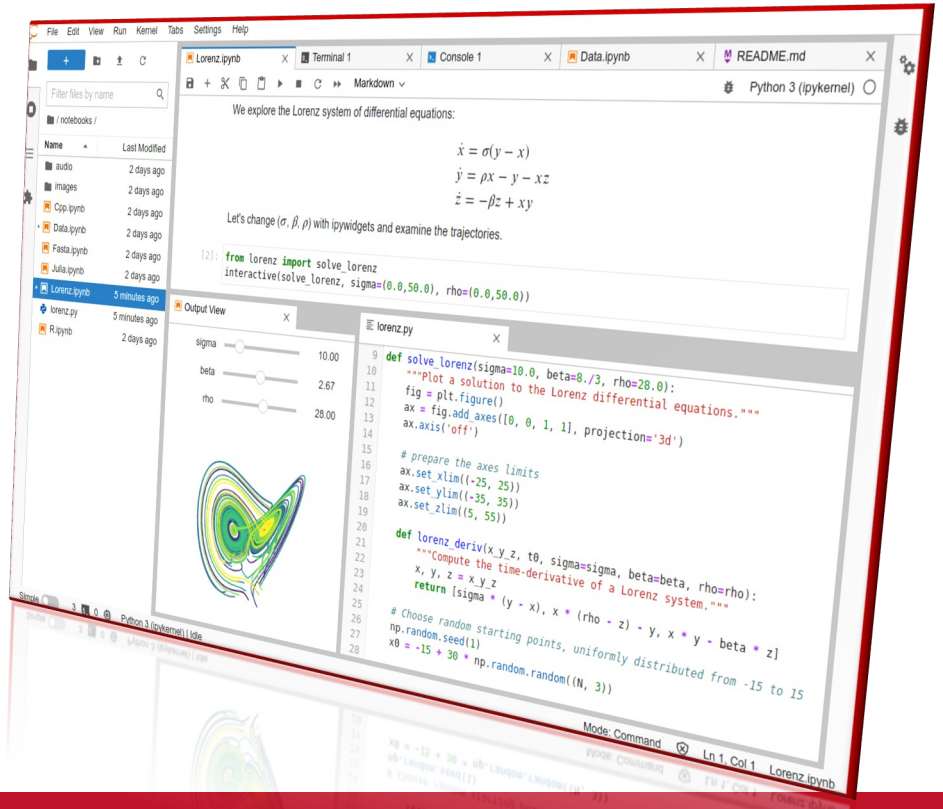
Laboratory setup

We will use your bootcamp setup!

ANACONDA® + jupyter

https://wiki.novasearch.org/wiki/lab_setup

Course Shared Folder



Carnegie Mellon Portugal | TÉCNICO LISBOA | Ciências ULisboa | NOVA SCHOOL OF SCIENCE & TECHNOLOGY

CMU Portugal Academy

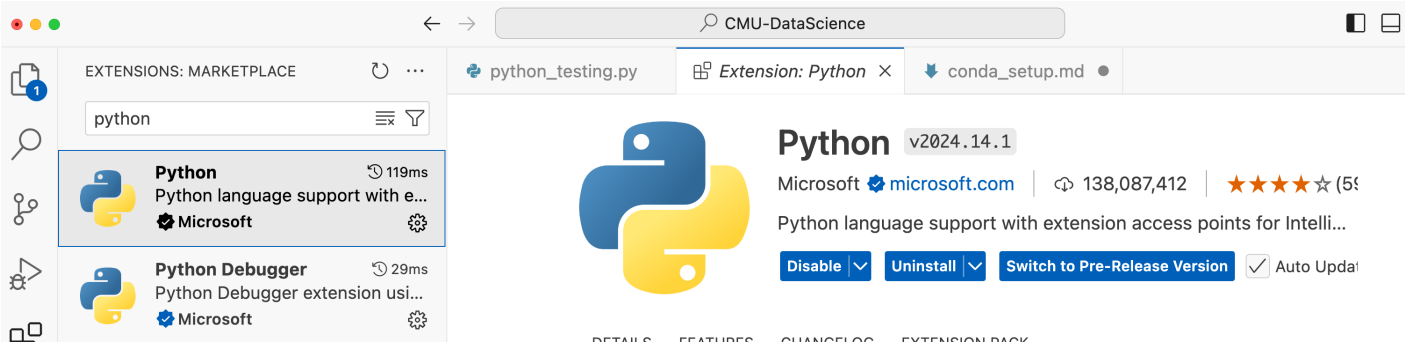
Laboratory setup – VS Code

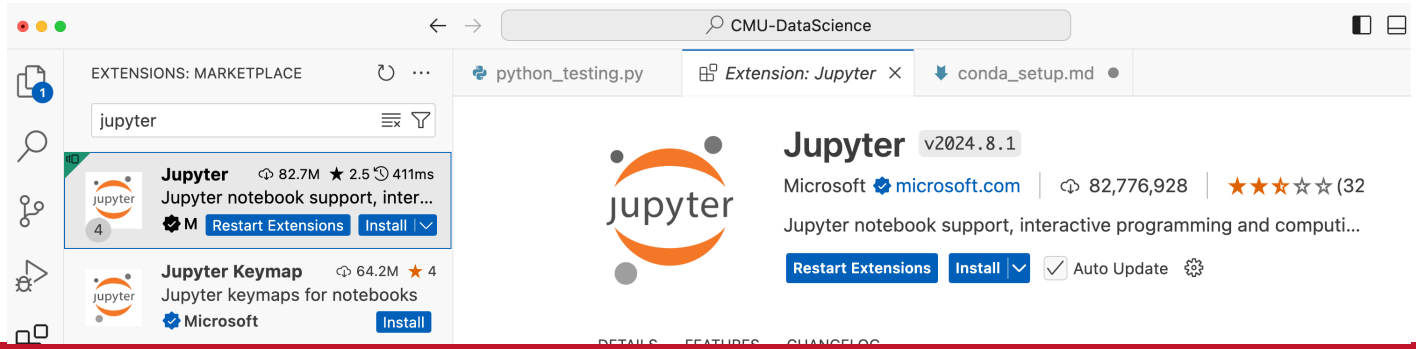
VS Code

Download VS Code

Install two extensions:

- Python
- Jupyter (Docs)





Carnegie Mellon Portugal · Técnico Lisboa · Ciências ULisboa · NOVA School of Science & Technology

CMU Portugal Academy

Course Use-Case

Apply Data Science tools to work with a specific use-case:

- Groups of 3 students
- To be done partially in lab sessions (on Saturdays)
- Final presentation in the last course session (18th October)

CMU Portugal Academy

Course Use-Case

Apply Data Science tools to work with a specific use-case.

Pick an interesting dataset:

- From Kaggle (Link) or UCI (Link)
- Bring your own

Requirements:

- Between 500 to 100k samples
- Tabular file (csv)
- Only categorical and/or numerical features

Bring your dataset next week to get our feedback!

CMU Portugal Academy

02

Intro to Python Programming

Carnegie Mellon Portugal · TÉCNICO LISBOA · Ciências ULisboa · NOVA SCHOOL OF SCIENCE & TECHNOLOGY

CMU Portugal Academy

Constants: Numbers

`float`
(floating point numbers)

`int` (Integer numbers)

```
In [1]: 1234
Out[1]: 1234

In [2]: -56
Out[2]: -56

In [3]: +3
Out[3]: 3

In [4]: 0
Out[4]: 0
```

```
In [1]: 3.64
Out[1]: 3.64

In [2]: -0.0747
Out[2]: -0.0747

In [3]: 2.0
Out[3]: 2.0

In [4]: 0.0
Out[4]: 0.0
```

CMU Portugal Academy

Constants: String

`str`
(String - sequence of characters between single or double quotation marks)

```
In [1]: "Hello, World!"
Out[1]: 'Hello, World!'

In [2]: 'Hello, World!'
Out[2]: 'Hello, World!'

In [3]: " Bom dia  :-) "
Out[3]: ' Bom dia :-) '

In [4]: "2 + (3 * 6)"
Out[4]: '2 + (3 * 6)'
```

Use double quotation marks!

CMU Portugal Academy

Constants: Function `type`

Constants have a **value** and a **type**.

```
In [1]: type(234)
Out[1]: int

In [2]: type(-1.24)
Out[2]: float

In [3]: type(3.0)
Out[3]: float

In [4]: type("Bom dia  :-)")
Out[4]: str
```

CMU Portugal Academy

Arithmetic expressions: operators

- Basic arithmetic operators:
  +, −, ∗, /, ∗∗, //, %

```
In [1]: 7 + 12 # addition
Out[1]: 19

In [2]: 6 / 2 # division (result is float)
Out[2]: 3.0

In [3]: 5 // 2 # integer division (result is int)
Out[3]: 2

In [4]: 3 ** 2 # exponentiation
Out[4]: 9

In [5]: 7 % 2 # division remainder
Out[5]: 1

In [6]: (5 / 2) * (6 - 3)
Out[6]: 7.5

In [7]: 2.3 + 4.6
Out[7]: 6.8999999999999995
```

Reference: https://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html

CMU Portugal Academy

Arithmetic expressions: result type

The result type of the evaluating an arithmetic expression depends on the operator and the types of the two operands

The result type is

- **int** if the operands are of type `int` and the operator is `+`, `-`, `*`, `**` , `//` ou `%`
- **float** in any other case.

CMU Portugal Academy

Type Conversions in Python

- `round(e)` rounds the value of `e` to the nearest `int`
- `int(e)` converts the value of `e` (string or number) to `int`
- `float(e)` converts the value of `e` (string or number) to `float`

```
In [1]: round(2.9)
Out[1]: 3

In [2]: int(2.9)
Out[2]: 2

In [3]: float(5)
Out[3]: 5.0

In [4]: float("45")
Out[4]: 45.0
```

CMU Portugal Academy

Operator precedence

How is the following expression evaluated?

```
In [1]: 5 + 2 ** 3 * 2
Out[1]: 21
```

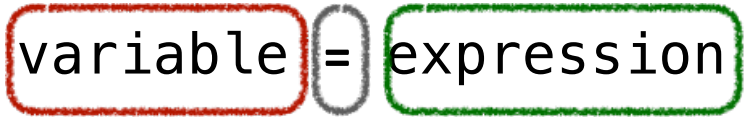
The following order (plus associativity) resolve ambiguity

Operator	Associativity	Precedence
( )		Highest
**	right	
– (symmetry)	right	
*, /, //, %	left	
+, –	left	Lowest

CMU Portugal Academy

Variables and assignment

- A variable is a symbolic name given to a **computer memory location** which is used to **store a value**.
- The assignment statement:



- First the expression is evaluated and then the association of the name to the value is made.
- Python keeps a record of the variables defined and their current values, this is called the *environment.*
- If the variable does not exist, it is added to the environment. Otherwise, its value is replaced by the value of the expression.

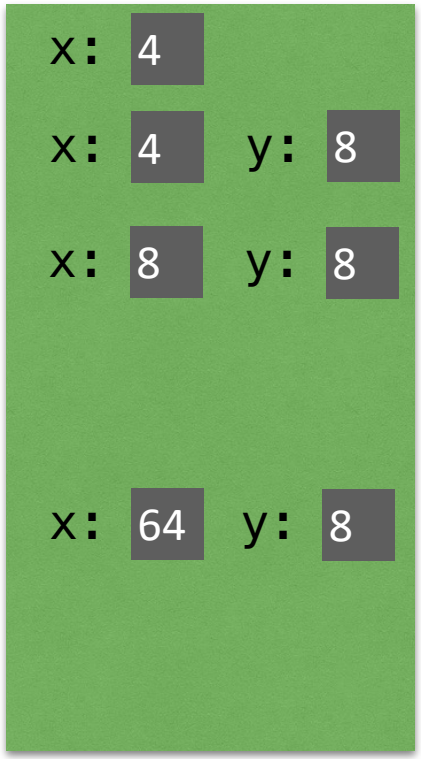
CMU Portugal Academy

Variables and binding

IPython console

```
In [1]: x = 4

In [2]: y = x * 2

In [3]: x = y

In [4]: x
Out[4]: 8

In [5]: x = x ** 2

In [6]: x
Out[6]: 64
```

Environment



Variable names

- The name of a variable is a sequence of alphanumeric characters and underscore, that starts with a letter or an underscore.
  - `grade`, `_max`, `minValue`, `name` ✔
  - `1teste` ✘

- Variable names are case-sensitive.
  - Names `Grade` and `grade` describe different variables.

CMU Portugal Academy

Variable names: Reserved keywords

- There is a set of Python keywords that cannot be used as variable names.

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

CMU Portugal Academy

Data input and output

- Function `input` reads a string from the terminal.
- To read numeric values we need to convert to the desired type.

```
In [1]: value = input("Amount to withdraw: ")
Amount to withdraw: 120

In [2]: type(value)
Out[2]: str

In [3]: value = int(input("Amount to withdraw: "))
Amount to withdraw: 120

In [4]: type(value)
Out[4]: int
```

CMU Portugal Academy

Data input and output

- The function `print` writes data to the terminal.

- This function does not return any value, it presents on the screen the result of the provided expression!

```
In [1]: print()

In [2]: print("Hello, World!")
Hello, World!

In [3]: print("2 + 4")
2 + 4

In [4]: print(2 + 4)
6

In [5]: print("2 + 4 = ", 2 + 4)
2 + 4 = 6
```

CMU Portugal Academy

Escape characters in Strings

- What if we wanted to write a String that has double quotes? For instance,

```
Alice said "I'm going for a coffee."
```

```
In [1]: print("Alice said "I'm going for a coffee."")
  File "<ipython-input-1-a4cbf5d481bb>", line 1
    print("Alice said "I'm going for a coffee."")
                       ^
SyntaxError: invalid syntax

In [2]: print("Alice said \"I'm going for a coffee.\"")
Alice said "I'm going for a coffee."
```

CMU Portugal Academy

Escape characters in Strings

What if we wanted to write a String that has double quotes? For instance,

Escape Sequence	Meaning
\’	Single quote
\\’	Double quote
\\	Backslash
\n	Newline
\r	Carriage Return
\t	Horizontal Tab
\b	Backspace
\f	Formfeed
\v	Vertical Tab
\0	Null Character
\Uxxxxxxxx	Unicode character with a 32-bit hex value
\000	Character with octal value ooo
\xhh	Character with hex value hh

Hands-On Mini-session

Let’s write a function that computes the net expected winnings from a bet, according to a given win probability, potential reward, and bet amount.

It should consider a 23% tax deduction on the winnings.

CMU Portugal Academy

Hands-On Mini-session

```python
probability = 0.7  # 70% chance of winning
reward_amount = 100  # 100€ reward
tax_rate = 0.23  # 23% tax

# Reading the bet amount from the user
bet_amount = float(input("Enter the bet amount: "))

# Calculate expected winnings before tax
expected_win = probability * reward_amount

# Apply tax to the expected winnings
expected_win_after_tax = expected_win * (1 - tax_rate)

net_winnings = expected_win_after_tax - bet_amount

print("Net expected winnings after tax and bet:" , round(net_winnings, 2))
```

CMU Portugal Academy

Booleans

- Beyond integers and floats, Python also has booleans (`bool`).
  - Only two constants: `True` and `False`

```
In [1]: True
Out[1]: True

In [2]: False
Out[2]: False

In [3]: type(True)
Out[3]: bool
```

`bool`
(logic value)

CMU Portugal Academy

The None value

Has everyone come across None/NULL/null/Nothing/nil/0?

The None keyword is used to define a null value, or no value at all.

```
In [12]: None

In [13]: print(None)
None

In [14]: type(None)
Out[14]: NoneType

In [15]: 4 + None
---------------------------------------------
TypeError   Traceback (most recent call last)
Cell In[15], line 1
----> 1 4 + None

TypeError: unsupported operand type(s) for +:
 'int' and 'NoneType'

In [16]:
```

CMU Portugal Academy

Logic operators

- Operators:

`and` (conjunction)	`or` (disjunction)	`not` (negation)

- Returns a boolean value from boolean values

```
In [1]: not True # negation
Out[1]: False

In [2]: True or False # disjunction
Out[2]: True

In [3]: True and False # conjunction
Out[3]: False

In [4]: not 0 # zero (int or float) is interpreted as False
Out[4]: True

In [5]: not 34 # any number not zero is interpreted as True
Out[5]: False
```

CMU Portugal Academy

Relational operators

== (equality)	!= (inequality)
< (less than)	<= (less or equal than)
> (greater than)	>= (greater or equal than)

- These operators return a boolean value from numeric values (`int` or `float`)

```
In [1]: x = 10  # assign value 10 to variable x

In [2]: x == 10 # checks if the value of variable x is equal to 10
Out[2]: True

In [3]: x >= 12 # checks if the value of variable x is greater or equal than
10
Out[3]: False

In [4]: x != 12 # checks if the value of variable x is not equal to 12
Out[4]: True
```

CMU Portugal Academy

Chaining relational operators

```
In [1]: x = 10  # assign value 10 to variable x

In [2]: 0 <= x <= 20 # checks if the value of x in in the interval [0,20]
Out[2]: True
```

`0 <= x <= 20` is equivalent to `0 <= x and x <= 20`

CMU Portugal Academy

Conditional statement: if-else

Syntax

```
if <exp_bool> :
    <instructions_true>
else :
    <instructions_false>
```

Example

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

- Evaluation steps
  1. Evaluate boolean expression `<exp_bool>`
     - If the value is **True** instructions `<instructions_true>` are executed.
     - If the value is **False** instructions `<instructions_false>` are executed.

CMU Portugal Academy

Chained conditional statements

Syntax

```
if <exp_bool_1> :
    <instrucoes_1>
elif <exp_bool_2> :
    <instrucoes_2>
else :
    <instrucoes_3>
```

Example

```
if x < y :
    print("x is less than y")
elif x > y :
    print("x is greater than y")
else :
    print("x is equal to y")
```

- The `else` can be omitted.

CMU Portugal Academy

Python Functions

- A function in Python is a sequence of instructions that is assigned a name.
- We can provide input parameters.
- Can return a result (or not)
- We can define our own functions.

```
def name(<parameters>):
    <instructions>
        [return <result>]
```

Hands-On Mini-session

Let’s rewrite our solution to the previous problem in a Function.

CMU Portugal Academy

Rewriting our solution to the previous problem in a Function.

```
probability = 0.7  # 70% chance of winning
reward_amount = 100  # 100€ reward
tax_rate = 0.23  # 23% tax

# Reading the bet amount from the user
bet_amount = float(input("Enter the bet amount: "))

# Calculate expected winnings before tax
expected_win = probability * reward_amount

# Apply tax to the expected winnings
expected_win_after_tax = expected_win * (1 - tax_rate)

net_winnings = expected_win_after_tax - bet_amount

print("Net expected winnings after tax and bet:" , round(net_winnings, 2))
```

CMU Portugal Academy

Rewriting our solution to the previous problem in a Function.

```python
def expected_reward(bet_amount):

    probability = 0.7  # 70% chance of winning
    reward_amount = 100  # 100€ reward
    tax_rate = 0.23  # 23% tax

    # Calculate expected winnings before tax
    expected_win = probability * reward_amount

    # Apply tax to the expected winnings
    expected_win_after_tax = expected_win * (1 - tax_rate)

    net_winnings = expected_win_after_tax - bet_amount

    return round(net_winnings, 2)
```

✓ 0.0s Python

```python
bet = float(input("Enter the bet amount: "))
result = expected_reward(bet)
print("Net expected winnings after tax and bet:" , result, "€")
```

✓ 3.0s Python

```
Net expected winnings after tax and bet: 23.9 €
```

Carnegie Mellon Portugal · TÉCNICO LISBOA · Ciências ULisboa · NOVA SCHOOL OF SCIENCE & TECHNOLOGY

CMU Portugal Academy

Functions and Local Variables

- A function has its own environment/scope.
- Local variables are destroyed after execution

```
    def power_twice(n, power):
        n_power = n**power
        return 2*n_power
[18]  ✓ 0.0s                                    Python

    power_twice(10, 2)
[20]  ✓ 0.0s                                    Python
...   200

    print(n_power)
[21]  ⊗ ✧ 0.0s                                  Python
...   ---------------------------------------------------------------
      NameError                                 Traceback
      Cell In[21], line 1
      ----> 1 print(n_power)

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

CMU Portugal Academy

Structuring programs with functions

- A program consists of multiple functions.
- Each concept should be programmed by a distinct function.
- Each function must have a name that clearly indicates the task it performs.
- Each function receives as arguments the data it needs to perform its task.

CMU Portugal Academy

Program Structure Good Practices

There are two types of functions: domain and interface.

Domain functions:

Implement the program logic.

Should not include user interaction (no input or print instructions).

I**nterface** functions:

Implement the interaction with the user.

Read data and present results.

Interface functions call the domain functions, but the opposite should not happen.

Do not mix function types - separate the logic from the user interface.

CMU Portugal Academy

Function `countdown`: first attempt

```
def countdown(counter):
    """countdown : int [0,3] -> str
    Description: build a string with a regressive counting from
      counter (a value between 0 and 3) to 1
    Examples: countdown(3) -> "3 2 1 GO!"; countdown(0) -> "GO!"
    """

    msg = ""
    if counter == 3:
        msg = msg + "3 "
    if counter >= 2:
        msg = msg + "2 "
    if counter >= 1:
        msg = msg + "1 "
    msg = msg + "GO!"
    return msg
```

```python
def countdown(counter):
    """countdown : int [0,100] -> str
    Description: build a string with a regressive counting from
      counter (a value between 0 and 100) to 1
    Examples: countdown(3) -> "3 2 1 GO!"; countdown(0) -> "GO!"
    """

    msg = ""
    if counter == 100:
        msg = msg + "100 "
    if counter >= 99:
        msg = msg + "99 "
    if counter >= 98:
        msg = msg + "98 "
       ...
    if counter >= 2:
        msg = msg + "2 "
    if counter >= 1:
        msg = msg + "1 "
    msg = msg + "GO!"
    return msg
```

CMU Portugal Academy

```
def countdown(counter):
    """countdown : int [0,100] -> str
    Description: build a string with a regressive counting from
      counter (a value between 0 and 100) to 1
    Examples: countdown(3) -> "3 2 1 GO!"; countdown(0) -> "GO!"
    """

    msg = ""
    if counter == 100:
        m   = msg      00 "
    if co     r >
        ms          "99 "
    if coun         98:
        msg         + "98 "
        ..
    if      er
        =  ms    "2 "
    if counter >= 1:
        msg = msg + "1 "
    msg = msg + "GO!"
    return msg
```

"Solution" **inadequate!**

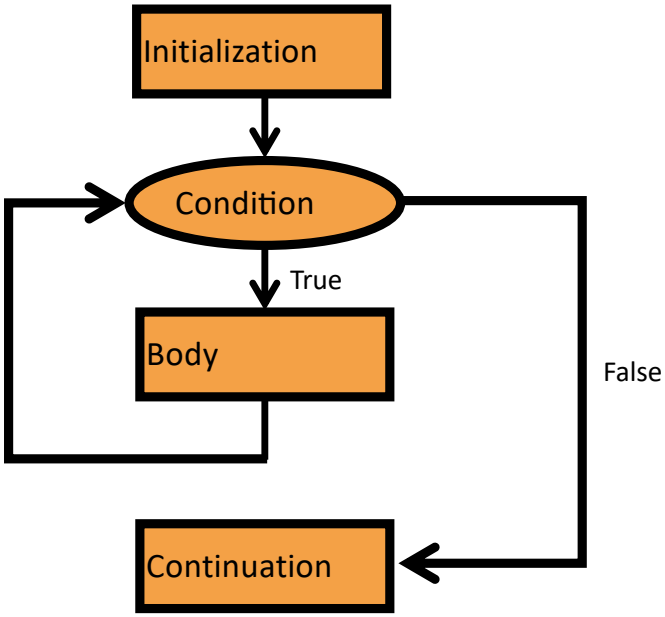
- Impractical long sequence of ifs.
- Arbitrary upper limit - **lacks generality**!

CMU Portugal Academy

While loop

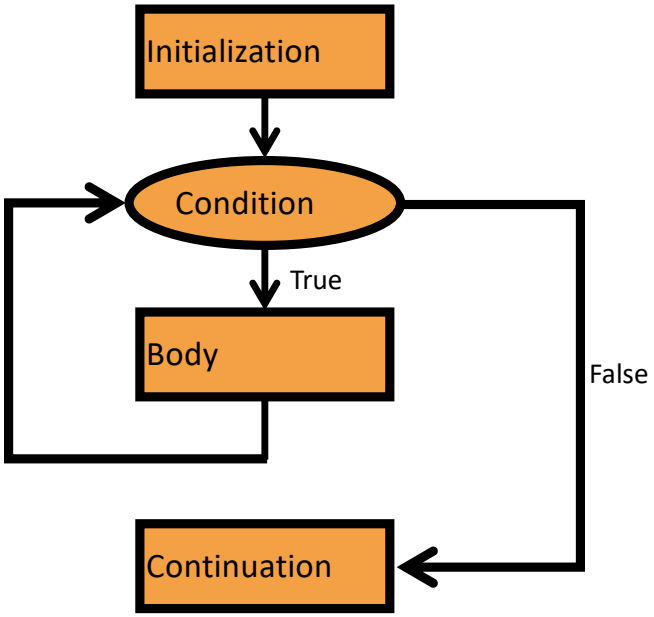
Syntax

```
<instructions_init>    # Initialization
while <condition> :    # Condition
   <instructions_body> # Body
<instructions_cont>    # Continuation
```



While loop

```
def countdown(counter):

    msg = ""

    while counter > 0:

        msg = msg + str(counter) + " "
        counter = counter - 1

    msg = msg + "GO!"
    return msg
```



If the condition is false the loop is not executed

If the execution enters the loop, the body should eventually render the condition false to allow exiting the loop

Hands-On Mini-session

Let’s write a function that checks if a positive integer number N is prime.

Function `is_prime`

How can we verify if a positive integer number is prime?

Function `is_prime`

How can we verify if a positive integer number is prime?

A simple (inefficient) algorithm is to check if it is divisible by any other number strictly between 1 and the given number we want to test.

Function `is_prime`

Let's implement the algorithm described as a function (is_prime) and test if it behaves as expected.

Write a program that accepts positive integer numbers given by the user and test them. To stop the program, the user should provide -1.

CMU Portugal Academy

Function `is_prime`

```
def is_prime(n):
    """is_prime : int [positive] -> bool
    Description: checks if n is prime.
    Examples: is_prime(3)->True; is_prime(4)->False
    """
    <initialization>
    while <condition>:
        <body>

    return <result>
```

CMU Portugal Academy

Function `is_prime`

```
def is_prime(n):
    """is_prime : int [positive] -> bool
    Description: checks if n is prime.
    Examples: is_prime(3)->True; is_prime(4)->False
    """
    <auxiliary_var_with_upper_limit>
    while <condition>:
        <body>

    return <result>
```

CMU Portugal Academy

Function `is_prime`

```
def is_prime(n):
    """is_prime : int [positive] -> bool
    Description: checks if n is prime.
    Examples: is_prime(3)->True; is_prime(4)->False
    """
    v = n - 1
    while <condition>:
        <body>

    return <result>
```

CMU Portugal Academy

Function `is_prime`

```
def is_prime(n):
    """is_prime : int [positive] -> bool
    Description: checks if n is prime.
    Examples: is_prime(3)->True; is_prime(4)->False
    """
    v = n - 1
    while <condition>:
        <body>

    return <did_v_reach_lower_limit>
```

CMU Portugal Academy

Function `is_prime`

```
def is_prime(n):
    """is_prime : int [positive] -> bool
    Description: checks if n is prime.
    Examples: is_prime(3)->True; is_prime(4)->False
    """
    v = n - 1
    while <condition>:
        <body>

    return v == 1
```

CMU Portugal Academy

Function `is_prime`

```
def is_prime(n):
    """is_prime : int [positive] -> bool
    Description: checks if n is prime.
    Examples: is_prime(3)->True; is_prime(4)->False
    """
    v = n - 1
    while <condition>:
        <decrement_v>

    return v == 1
```

CMU Portugal Academy

Function `is_prime`

```
def is_prime(n):
    """is_prime : int [positive] -> bool
    Description: checks if n is prime.
    Examples: is_prime(3)->True; is_prime(4)->False
    """
    v = n - 1
    while <condition>:
        v = v - 1

    return v == 1
```

CMU Portugal Academy

Function `is_prime`

```
def is_prime(n):
    """is_prime : int [positive] -> bool
    Description: checks if n is prime.
    Examples: is_prime(3)->True; is_prime(4)->False
    """
    v = n - 1
    while <n_not_divisible_by_v>:
        v = v - 1

    return v == 1
```

CMU Portugal Academy

Function `is_prime`

```
def is_prime(n):
    """is_prime : int [positive] -> bool
    Description: checks if n is prime.
    Examples: is_prime(3)->True; is_prime(4)->False
    """
    v = n - 1
    while n % v != 0:
        v = v - 1

    return v == 1
```

CMU Portugal Academy

Function `is_prime`

```
def is_prime(n):
    """is_prime : int [positive] -> bool
    Description: checks if n is prime.
    Examples: is_prime(3)->True; is_prime(4)->False
    """
    v = n - 1
    while n % v != 0:
        v = v - 1

    return v == 1
```

What if n is not positive?

CMU Portugal Academy

Function `is_prime`

```
def is_prime(n):
    """is_prime : int [positive] -> bool
    Description: checks if n is prime.
    Examples: is_prime(3)->True; is_prime(4)->False
    """
    v = n - 1
    while (v > 1) and (n % v != 0):
        v = v - 1

    return v == 1
```

CMU Portugal Academy

Testing our `is_prime` implementation

```python
def is_prime(n):
    """is_prime : int [positive] -> bool
    Description: checks if n is prime.
    Examples: is_prime(3)->True; is_prime(4)->False
    """
    v = n - 1
    while (v > 1) and (n % v != 0):
        v = v - 1

    return v == 1
```

[24] ✓ 0.0s Python

```python
print(is_prime(7), is_prime(123), is_prime(1009))
```

[28] ✓ 0.0s Python

```
True False True
```

CMU Portugal Academy

For Loop

Instruction `for` repeats a block of statements for each element of the sequence.

Syntax

```
for <variable> in <sequence>:
   <instruction>
```

Example

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

- If variable **i** does not exist, it is created.
- For each element **e** of sequence **s** (following the order of the elements in the sequence):
  1. **i = e**;
  2. Then execute the block of instructions within the loop (in the example, instruction `print`)

CMU Portugal Academy

For Loop: Example

- Sequence defined by function `range`.

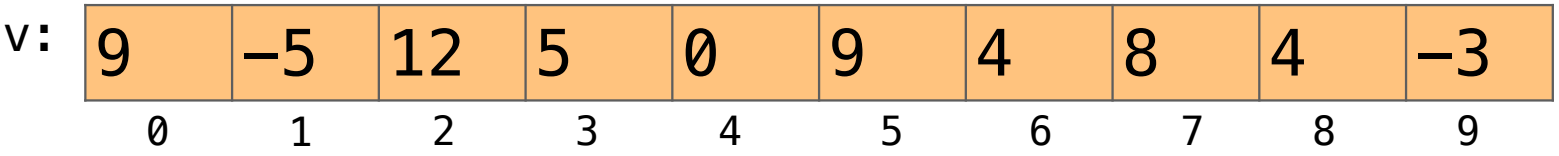
```
In [1]: for i in range(2, 6):
   ...:     print(i)
   ...:
2
3
4
5
```

CMU Portugal Academy

Lists in Python

- Create a list and assign it to variable v:

```
v = [9, -5, 12, 5, 0, 9, 4, 8, 4, -3]
```



- Function `len` returns the size of v: `len(v)` is 10
- Position 3 of v: `v[3]` is 5

Updating a position (list)

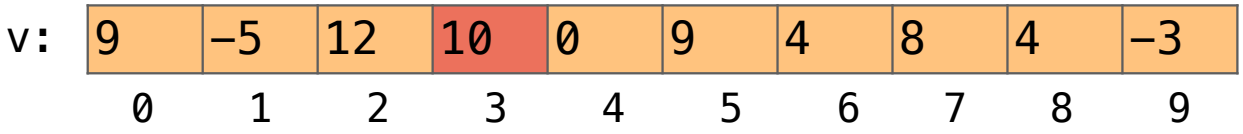
Syntax for updating the contents of a position:

- `variable_name[`**expression**`]` = **expression**

```
v[3] = 10
```

```
v[1+3] = -2
```

```
if k==8, v[k] = 0
```

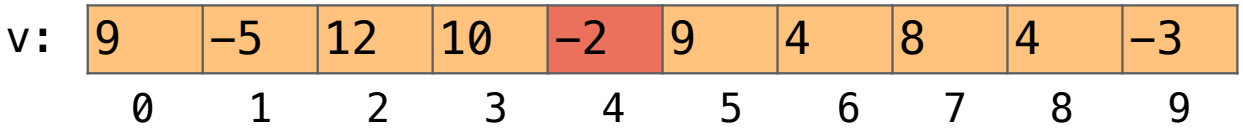


Updating a position (list)

Syntax for updating the contents of a position:

- `variable_name[`**expression**`]` = **expression**

```
v[3] = 10
v[1+3] = -2
if k==8, v[k] = 0
```



CMU Portugal Academy

Typical indexing errors

```
v = [9, -5, 12, 5, 0, 9, 4, 8, 4, -3]

v[-1] = 9                    # The index cannot be negative

x = v[ math.sqrt(4) ]        # The index has to be an integer

y = v[12]                    # The index has to be ≤ len(v)-1

v[12] = 4                    # The index has to be ≤ len(v)-1
```

CMU Portugal Academy

Slicing Lists

Lists can be sliced:

- *To slice elements within a range start_index and end_index (inclusive), we can use [start_index : end_index+1]*

A	B	C	D	F
0	1	2	3	4

```
>>> grades = ["A", "B", "C", "D", "F"]
>>> grades[1:3]
['B', 'C']
>>>
```

CMU Portugal Academy

Slicing Lists

Lists can be sliced:

- *To slice elements from the beginning to a certain ending index (inclusive), we can use [: end_index+1]*

A	B	C	D	F
0	1	2	3	4

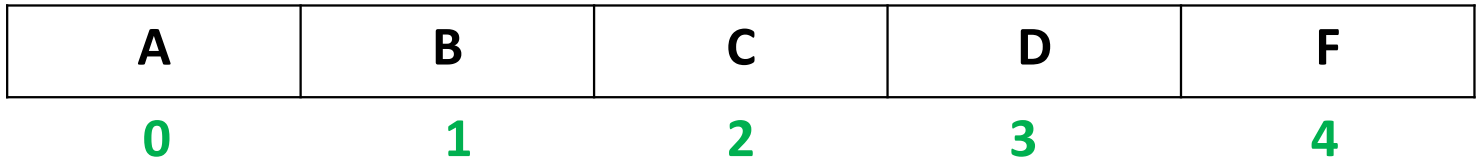
```
>>> grades = ["A", "B", "C", "D", "F"]
>>> grades[:3]
['A', 'B', 'C']
>>>
```

CMU Portugal Academy

Slicing Lists

Lists can be sliced:

- *To slice elements from a certain starting index till the end, we can use [start_index:]*

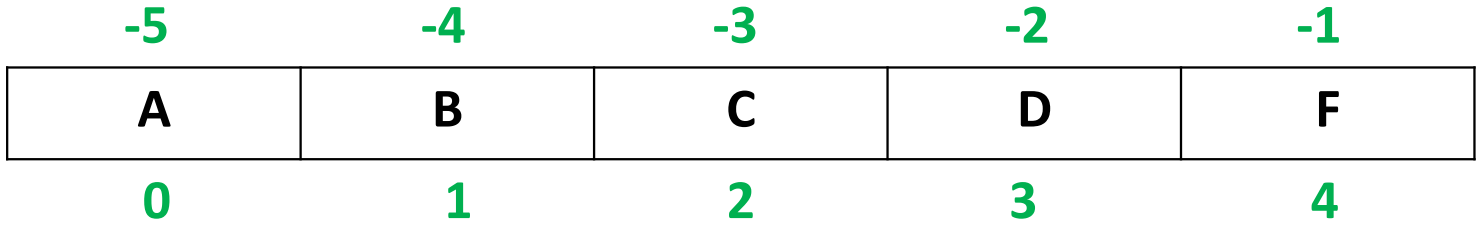



```
>>> grades = ["A", "B", "C", "D", "F"]
>>> grades[1:]
['B', 'C', 'D', 'F']
>>>
```

Slicing Lists

Lists can be sliced:

- *We can even pass a negative index, after which Python will add the length of the list to the index*



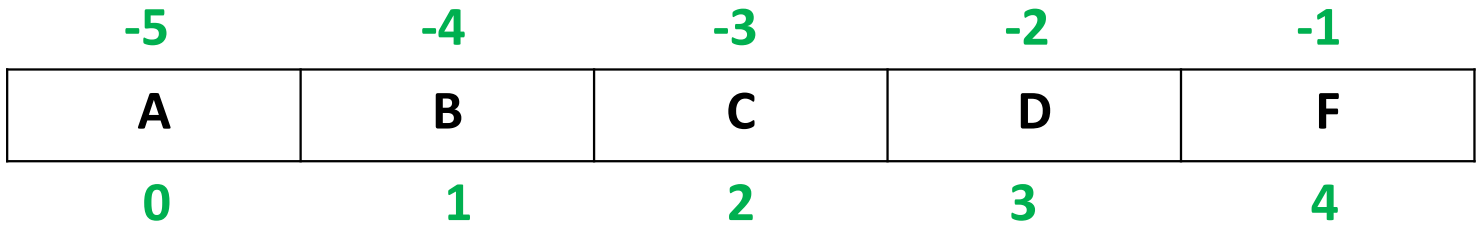

```
>>> grades = ["A", "B", "C", "D", "F"]
>>> grades[-1]
'F'
>>>
```

CMU Portugal Academy

Slicing Lists

Lists can be sliced:

- *To slice elements from a certain ending index (inclusive) till the beginning, we can use [:-end_index+1]*



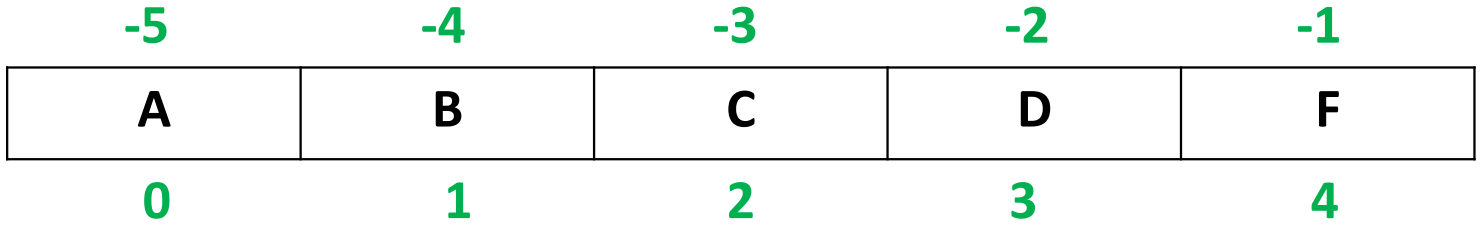

```
>>> grades = ["A", "B", "C", "D", "F"]
>>> grades[:-1]
['A', 'B', 'C', 'D']
>>>
```

CMU Portugal Academy

Slicing Lists

Lists can be sliced:

- *To slice in steps, we can use [start_index, end_index+1, step] (idea applies to slicing from forward and backward)*




```
>>> grades = ["A", "B", "C", "D", "F"]
>>> grades[0:5:2]
['A', 'C', 'F']
>>>
```

CMU Portugal Academy

Lists Are Not Sets

- Lists can have *duplicate* values

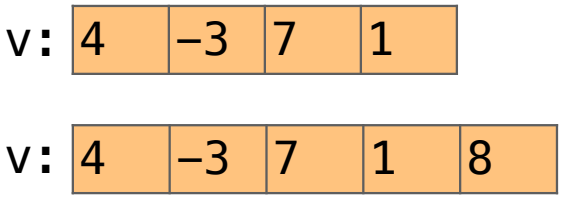
```
>>> myList = [5, 2, 2, 3]
>>> myList
[5, 2, 2, 3]
>>>
```

CMU Portugal Academy

List: Functions `append` and `extend`

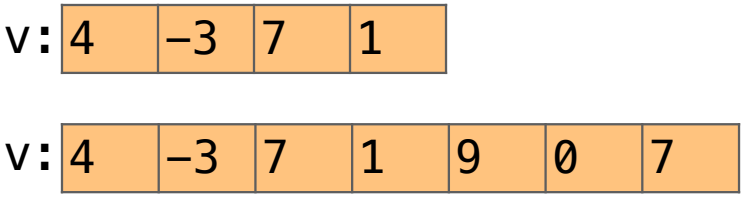
append adds an element to the end of the list:

```
>>> v = [4, -3, 7, 1]
>>> v.append(8)
```



`extends` adds a sequence of elements to the end of the list:

```
>>> v = [4, -3, 7, 1]
>>> v.extend([9, 0, 7])
```



CMU Portugal Academy

Python Reference – Lists

List Functions

- Here is the set of functions that you can use with lists

Function	Description
L.append(x)	Add element x to the end of list L
L.extend(L2)	Add all elements of list L2 to the end of list L
L.insert(i, x)	Insert item x at the defined index i of list L
L.remove(x)	Removes item x from list L (valueError exception will be thrown if x does not exist)
L.pop(i)	Removes and returns the element at index i of list L. If no parameter is passed, the last item in L will be removed and returned

CMU Portugal Academy

Python Reference – Lists

List Functions

- Here is the set of functions that you can use with lists

Function	Description
L.clear()	Removes all items from list L
L.index(x)	Returns the index of the first matched item x in list L
L.count(x)	Returns the count of times item x appears in list L
L.sort()	Sort items in list L in an ascending order
L.reverse()	Reverses the order of items in list L
L.copy()	Returns a copy of list L (*making any change to the returned list will not impact the original list L*)

CMU Portugal Academy

Lists: Useful functions

- `v.index(n)` returns the index of the first occurrence of `n` in `v`
- `v.remove(n)` removes the first occurrence of value `n` in `v`
- `v.reverse(n)` inverts `v` (reverses the order of elements)
- `v.count(n)` counts the occurrences of value `n` in `v`

```
In  [1]: v = [10, 11, 12, 11, 14]
In  [2]: v.index(11)
Out [2]: 1
In  [3]: v.count(11)
Out [3]: 2
```

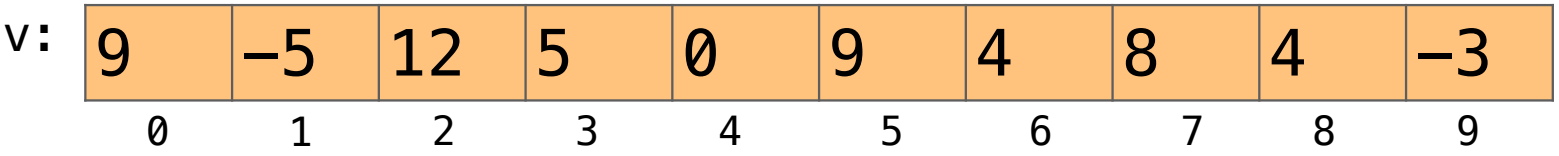
```
In  [4]: v.remove(11)
In  [5]: v
Out [5]: [10, 12, 11, 14]
In  [6]: v.reverse()
In  [7]: v
Out [7]: [14, 11, 12, 10]
```

CMU Portugal Academy

Tuples in Python – Immutable!

- Create a list and assign it to variable v:

```
v = (9, -5, 12, 5, 0, 9, 4, 8, 4, -3)
```



- Function `len` returns the size of v: `len(v)` is 10
- Position 3 of v: `v[3]` is 5
- We can't change the initial values.

Carnegie Mellon Portugal · TÉCNICO LISBOA · Ciências ULisboa · NOVA SCHOOL OF SCIENCE & TECHNOLOGY

CMU Portugal Academy

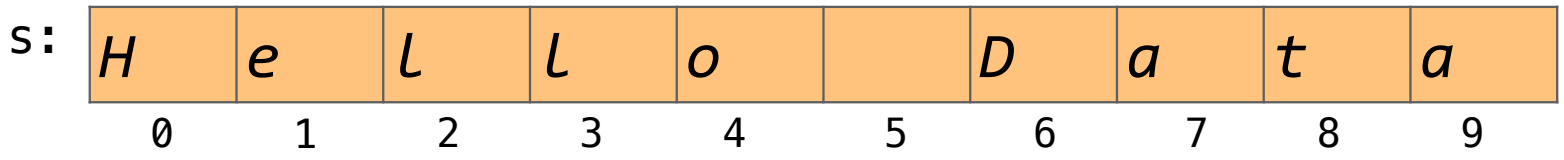
Tuples

- Like lists, tuples can:
  - Contain any and different types of elements
  - Contain duplicate elements (e.g., (1, 1, 2))
  - Be indexed exactly in the same way (i.e., using the [] brackets)
  - Be sliced exactly in the same way (i.e., using the [::] notation)
  - Be concatenated (e.g., t = (1, 2, 3) + (“a”, “b”, “c”))
  - Be repeated (e.g., t = (“a”, “b”) * 10)
  - Be nested (e.g., t = ((1, 2), (3, 4), ((“a”, “b”, ”c”), 3.4))
  - Be passed to a function, but will result in *pass-by-value* and not *pass-by-reference* outcome since it is immutable
  - Be iterated over

Strings: Sequences of characters

- Create a string and assign it to variable `s`:

```
s = "Hello Data"
```



- Function `len` returns the length of `s`: `len(s)` is 10
- Position 2 of `s`: `s[2]` is `"l"`
- Cannot update the contents of a position of `s`.

CMU Portugal Academy

For loop: Example

Iterating over the elements of a String

```
In [1]: s = "Hello!"
In [2]: for i in s:
   ...:     print(i)
   ...:
H
e
l
l
o
!
```

CMU Portugal Academy

Towards Dictionaries

- Lists and tuples hold elements with *only integer* indices

45	"Coding"	4.5	7	89
0	1	2	3	4

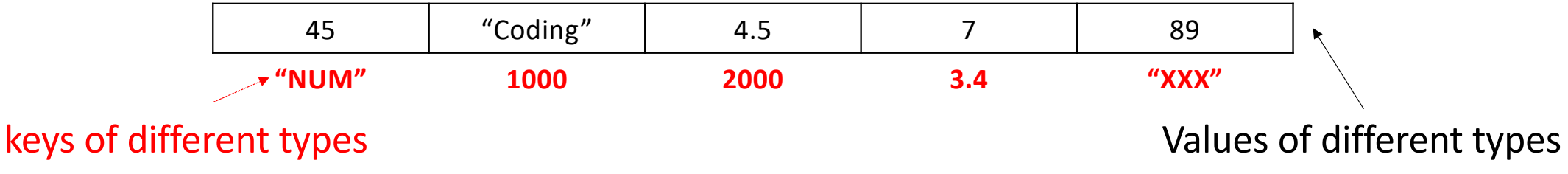
Integer Indices

- So in essence, each element has an *index* (or a *key*) which can *only* be an integer, and a *value* which can be of any type (e.g., in the above list/tuple, the first element has key 0 and value 45)
  - *What if we want to store elements with non-integer indices (or keys)?*

CMU Portugal Academy

Dictionaries

- We can use a dictionary to store elements with *keys of any types* and *values of any types* as well



- The above dictionary can be defined in Python as follows:

```
dic = {"NUM":45, 1000:"coding", 2000:4.5, 3.4:7, "XXX":89}
```

key *value*

key:*value* pairs separated by commas

CMU Portugal Academy

Dictionaries

- Can contain any and types of elements (i.e., keys and values)
- Can contain only *unique* different keys but duplicate values

```
dic2 = {"a":1, "a":2, "b":2}
print(dic2)
```

Output: {'a': 2, 'b': 2}

The element “a”:2 will override the element “a”:1 because only ONE element can have key “a”

- Can be indexed *but only* through keys (i.e., dic2[“a”] will return 1
- dic2[0] will return an error since there is no element with key 0 in dic2 above)

CMU Portugal Academy

Accessing an element

Python syntax to get the value associated with a **key**:

```
In [1]: balance = {"alice" : 12, "bob" : 23, "anne" : 12}
In [2]: type(balance)
Out[2]: dict
In [3]: balance["bob"]
Out[3]: 23
In [4]: balance["mary"]
Out[4]: Traceback (most recent call last):
  File "<ipython-input-19-cfcb43ec7e06>", line 1, in <module>
    balance["mary"]
KeyError: 'mary'
```

CMU Portugal Academy

Accessing an element

Updating an element. If the element is in the dictionary, its value is updated. Otherwise, a new **key**:**value** pair is added to the dictionary.

```
In [1]: balance = {"alice" : 12, "bob" : 23, "anne" : 12}
In [2]: balance["mary"] = 20
In [3]: balance
Out[3]: {'alice': 12, 'bob': 23, 'anne': 12, 'mary': 20}
In [4]: balance["alice"] = 20
In [5]: balance
Out[5]: {'alice': 20, 'bob': 23, 'anne': 12, 'mary': 20}
In [6]: balance["mary"] = balance["mary"] + 4
In [7]: balance
Out[7]: {'alice': 20, 'bob': 23, 'anne': 12, 'mary': 24}
```

CMU Portugal Academy

Operations over dictionaries

Check if a **key** is in the dictionary:

```
In [1]: balance = {"alice" : 12, "bob" : 23, "anne" : 12}
In [2]: "alice" in balance
Out[2]: True
In [3]: "john" in balance
Out[3]: False
In [4]: "john" not in balance
Out[4]: True
```

CMU Portugal Academy

Operations over dictionaries

pop removes the element associated to the key and returns the value.

```
In [1]: balance = {"alice" : 12, "bob" : 23, "anne" : 12}
In [2]: balance.pop("alice")
Out[2]: 12
In [3]: balance
Out[3]: {"bob" : 23, "anne" : 12}
In [4]: balance["alice"]
Out[4]: Traceback (most recent call last):
  File "<ipython-input-19-e74b4fdfdea2>", line 1, in <module>
    balance["alice"]
KeyError: 'alice'
```

CMU Portugal Academy

Operations over dictionaries

FOR Iterates over the **keys** of a dictionary:

```
In [1]: balance = {"alice" : 12, "bob" : 23, "anne" : 12}
In [2]: for key in balance:
   ...:     print(key)
   ...:
alice
bob
anne
```

CMU Portugal Academy

Views over a dictionary

- **keys()** returns a view over the keys.
- **values()** returns a view over the values.
- **items()** returns a view over the pairs key:value.

```
In [1]: balance = {"alice" : 12, "bob" : 23, "anne" : 12}
In [2]: balance.keys()
Out[2]: dict_keys(['alice', 'bob', 'anne'])
In [3]: balance.values()
Out[3]: dict_values([12, 23, 12])
In [4]: balance.items()
Out[4]: dict_items([('alice',12), ('bob',23), ('anne',12)])
```

CMU Portugal Academy

```
balance = {"alice" : 12, "bob" : 23, "anne" : 12 }
```

```
In [1]: for k in balance.keys():
   ...:     print(k)
   ...:
alice
bob
anne
```

```
In [1]: for k in balance.values():
   ...:     print(k)
   ...:
12
23
12
```

```
In [1]: for k, v in balance.items():
   ...:     print(k, v)
   ...:
alice 12
bob 23
anne 12
```

CMU Portugal Academy

Views over a dictionary

```
In [1]: balance = {"alice" : 12, "bob" : 23, "anne" : 12}

In [2]: list_keys = list(balance.keys())

In [3]: list_keys
Out[3]: ['alice', 'bob', 'anne']

In [4]: list_values = list(balance.values())

In [5]: list_values
Out[5]: [12, 23, 12]

In [6]: list_items = list(balance.items())

In [7]: list_items
Out[7]: [('alice', 12), ('bob', 23), ('anne', 12)]
```

CMU Portugal Academy

Dictionary Functions

- Many other functions can also be used with dictionaries

Function	Description
dic.clear()	Removes all the elements from dictionary dic
dic.copy()	Returns a copy of dictionary dic
dic.items()	Returns a list containing a tuple for each key-value pair in dictionary dic
dic.get(k)	Returns the value of the specified key k from dictionary dic
dic.keys()	Returns a list containing all the keys of dictionary dic
dic.pop(k)	Removes the element with the specified key k from dictionary dic

CMU Portugal Academy

Dictionary Functions

- Many other functions can also be used with dictionaries

Function	Description
dic.popitem()	Removes the last inserted key-value pair in dictionary dic
dic.values()	Returns a list of all the values in dictionary dic

Hands-On Session!

Course Shared Folder

CMU Portugal Academy

Carnegie Mellon Portugal · TÉCNICO LISBOA · Ciências ULisboa · NOVA SCHOOL OF SCIENCE & TECHNOLOGY

CMU Portugal Academy

This slides contain adaptations from the following references:

- Advanced Programming for Biology Course, FCT-NOVA
- 15-110: Principles of Computing course, CMU

CMU Portugal Academy

03

Numerical Operations

Carnegie Mellon Portugal | TÉCNICO LISBOA | Ciências ULisboa | NOVA SCHOOL OF SCIENCE & TECHNOLOGY

CMU Portugal Academy

Carnegie Mellon Portugal · TÉCNICO LISBOA · Ciências ULisboa · NOVA SCHOOL OF SCIENCE & TECHNOLOGY