Développer l'autonomie des élèves

Février 2024

Introduction

Supports

Supports

- Avec un IDE
- Avec un Notebook
- Avec Capytale

Objectifs

Objectifs

Préparer des supports permettant une vérification automatique du code produit par les élèves (tests préparés et fournis par l'enseignant dans un fichier séparé).

- Respect du cahier des charges
- Respect des bonnes pratiques

Les élèves peuvent ainsi en autonomie (ou pas) vérifier leur code et le corriger, étape par étape.

Supports : IDE / Notebook / Capytale.

Test présentés

Test présentés

- **Variables** : existence, valeur, type
- **Fonctions** : existence, arguments, valeurs de retour, préconditions
- **Classes** : existence, attributs (cf variables), méthodes (cf fonctions)

Les outils

doctest()

Principe

- Module python, effectue les tests des docstrings.
- Documentation [1]

[1] https://docs.python.org/fr/3/library/doctest.html

run_docstring_examples I

```python
def inc(x):
    """ Fonction qui ajoute 2.
    :param x: un nombre entier
    :return: le nombre+2
    ----------
    >>> inc(4)
    6
    >>> inc(-2)
    0
    """
    return x+2
```

run_docstring_examples II

```
from doctest import run_docstring_examples
run_docstring_examples(inc, globals(), verbose=True)
```

testmod I

```
"""
Documentation du module.

>>> inc(4)
6
"""
# code, fonctions (...)
if __name__ == "__main__":
    import doctest
    doctest.testmod(verbose=True)
```

Syntaxe try…except…else

Principe

Permet d'essayer un code (try) et d'exécuter une instruction différente si une exception est levée (ce qui signifie que le code a échoué - instruction except) ou non (else)

Le type de l'exception peut être précisé dans le `except` (par exemple `except ValueError`) de façon à ne pas intercepter toutes les exceptions.

Exemple : variable

```
x=0
try :
    1/x
except DivisionByZero:
    print("Nope, on ne peut pas diviser par 0.")
else:
    print("OK, opération possible. " )
```

Exemple : précondition

```
def div(a,b):
    assert b!=0
    return a/b
# Vérification
try:
    div(4,0)
except DivisionByZero:
    print("[err] Vérification dénominateur non nul.")
except AssertionError:
    print("[ok] Vérification dénominateur non nul.")
```

Récupération du message de l'exception

```
try:
    assert False, "ERREUR NORMALE"
except AssertionError as e:
    if e.args[0] == "ERREUR NORMALE":
        print("Tout va bien !")
```

Module inspect

Module inspect

- Module python, permet d'obtenir des informations sur les objets (modules, classes, fonctions, méthodes, etc.)
- Documentation [2]

Exemples d'utilisation :

- `inspect.getsource` : récupérer le code source d'un module / fonction / ...
- `inspect.signature` pour récupérer la signature d'une fonction.

[2] https://docs.python.org/fr/3/library/inspect.html

inspect.signature

`inspect.signature(fonction)` : renvoie un objet `Signature` qui contient les informations sur les paramètres (`inspect.Parameter`) de la fonction / méthode.

- `sig.parameters` : dictionnaire *ordonné* des paramètres
- `Parameter` : nom, valeur par défaut, type..

Exemple : signature de fonction

```
def diff(a, b=3):
    return a-b

import inspect
sig = inspect.signature(diff)
print(list(sig.parameters.keys())) # ['a', 'b']
sig.parameters["a"].default == inspect._empty
sig.parameters["b"].default == 3
sig.parameters.values()[1].name == "b"
```

getattr et hasattr

getattr et hasattr

- `getattr(objet, "attribut")` : renvoie la valeur de l'attribut de l'objet (et lève une exception si l'attribut n'existe pas)
- `getattr(objet, "attribut", defaut)` : renvoie la valeur de l'attribut de l'objet (ou la valeur par défaut si l'attribut n'existe pas)
- `hasattr(objet, "attribut")` : renvoie `True` si l'attribut existe, `False` sinon

Documentation : doc getattr [3] doc hasattr [4]

[3] https://docs.python.org/fr/3/library/functions.html#getattr
[4] https://docs.python.org/fr/3/library/functions.html#hasattr

Exemple

```
class Point:
    def __init__(self, x):
        self.x = x
    def move(self, dx):
        self.x += dx
p = Point(3)
hasattr(p, "x") # True
hasattr(p, "move") # True
getattr(p, "x") # 3
getattr(p, "move") # méthode
getattr(p, "bouge", None) # None
getattr(p, "bouge") # AttributeError
```

Tester un "print"

Redirection de la sortie standard

- `sys.stdout` : sortie standard
- `sys.stderr` : sortie d'erreur

Exemple

```
from contextlib import redirect_stdout
from io import StringIO
out = StringIO()
with redirect_stdout(f):
    print("Hello")
print(out.getvalue()) # "Hello\n"
```

Mise en place

Concept

Tests dans un fichier séparé

- Les tests sont écrits dans un fichier séparé, par exemple `_tests.py`
- Les tests sont écrits dans des fonctions, par exemple `test_exo1()`, `test_exo2()`
- L'élève n'a qu'à importer le fichier de tests au début de son code, et lancer les tests à chaque étape.

Exemple de fichier élève

```
import _tests as tests

# Exercice 1
# ...
tests.test_exo1()
```

Avec un IDE

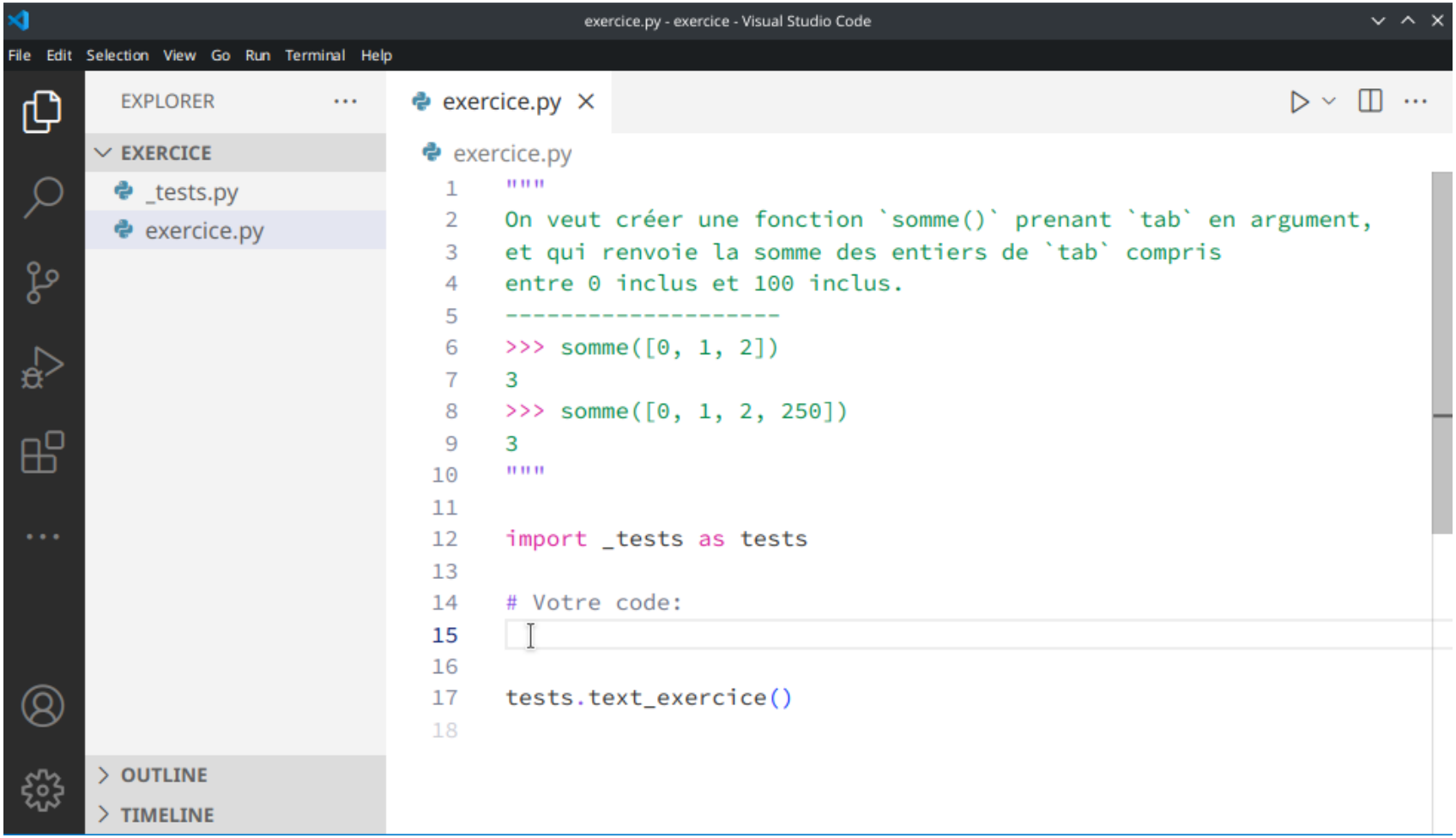
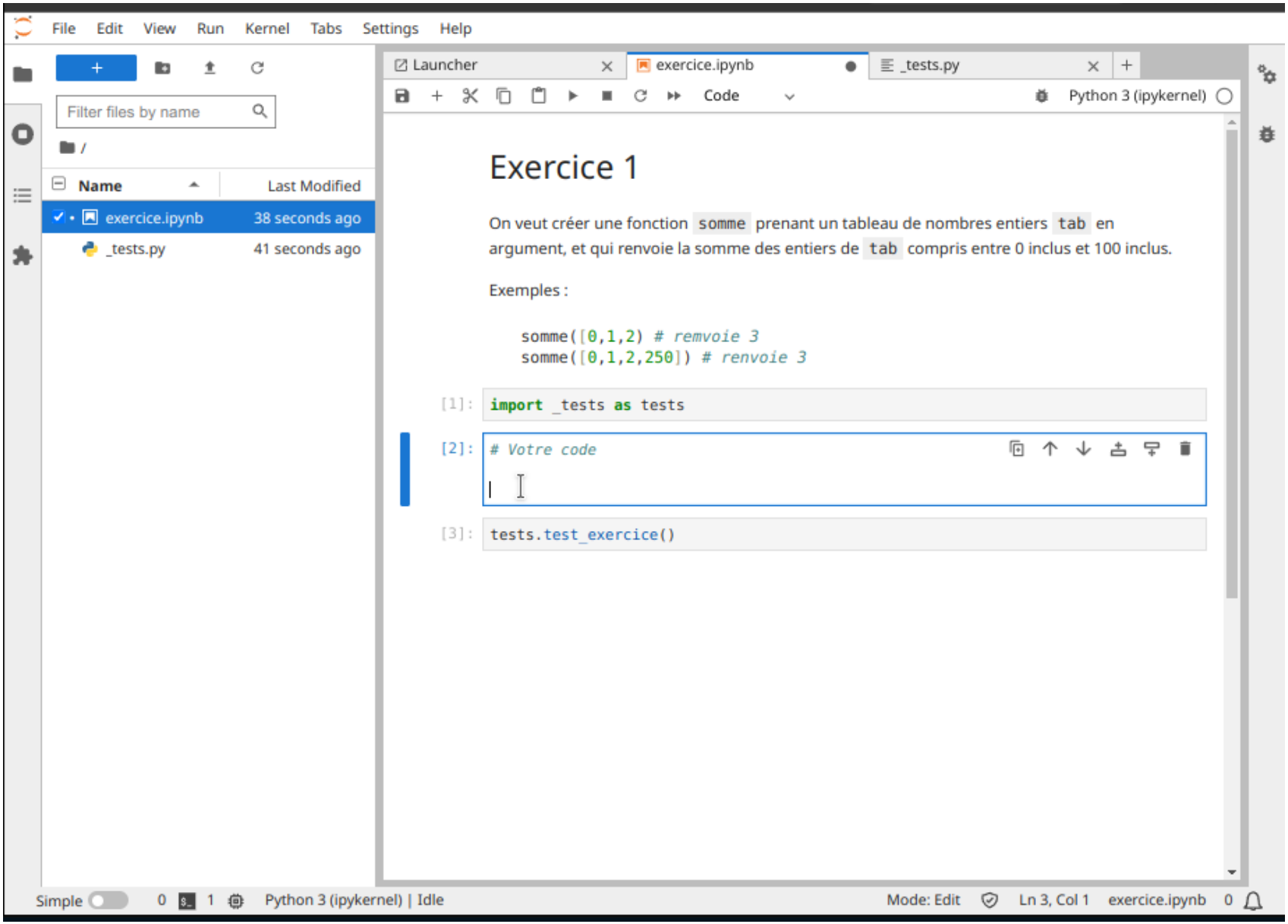


Figure 1: IDE

Avec un Notebook



Avec Capytale

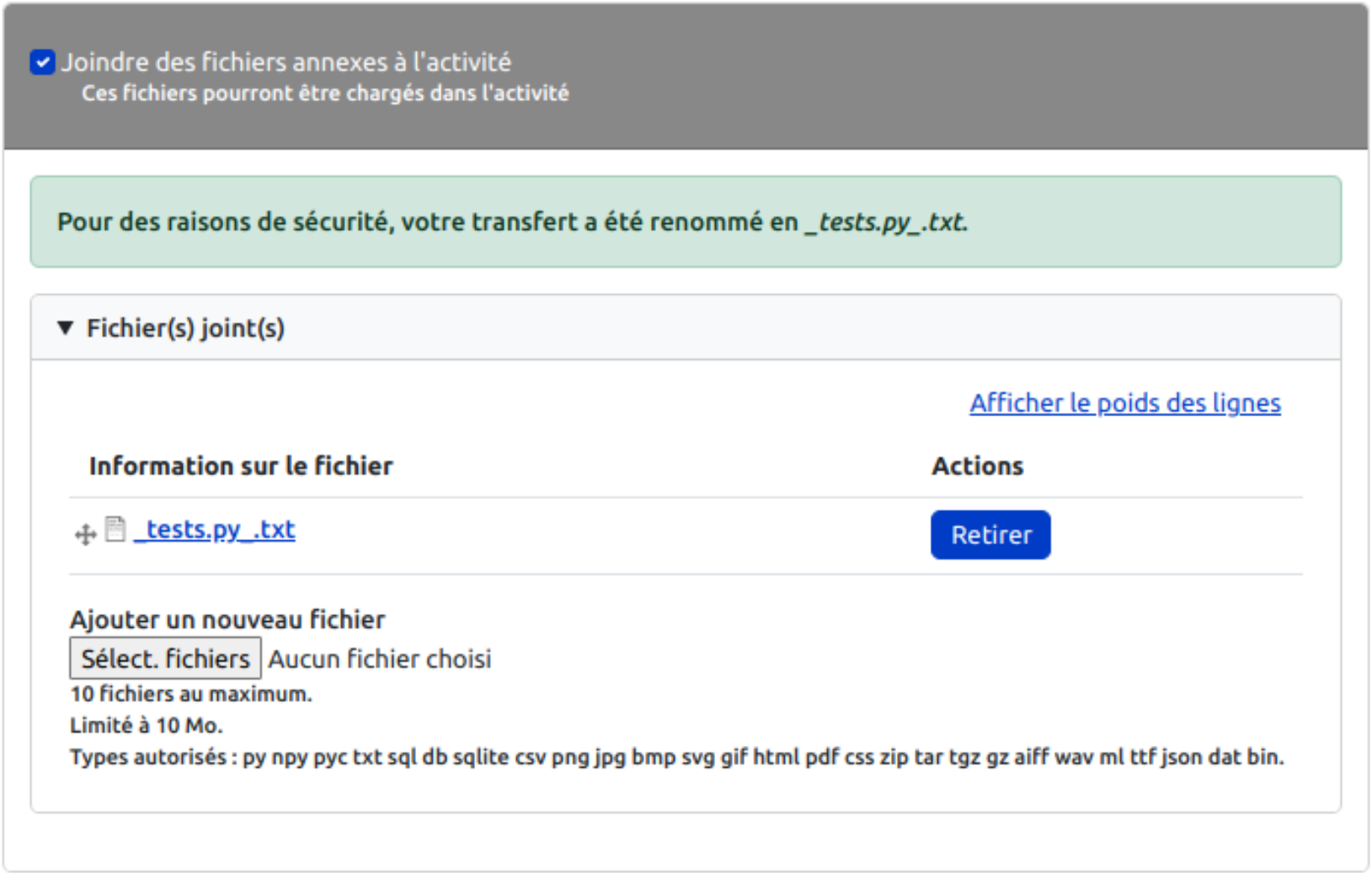


Figure 3: Capytale

Problème

Le module de tests doit pouvoir accéder aux variables, fonctions, classes définies par l'élève.

Solution : `inspect` encore..

```
def test_exo1():
    # Magique...
    g = inspect.stack()[1].frame.f_globals
```

inspect.stack() renvoie une liste de `FrameInfo` [5] (environnements d'exécution), le premier élément est l'environnement courant, le second est celui de l'élève. `.frame` [6] renvoie l'objet `frame` correspondant, et `.f_globals` renvoie ses variables globales.

---

[5] https://docs.python.org/fr/3/library/inspect.html#inspect.FrameInfo

[6] ou `[0]` avant python 3.5

Exemples

Variable

Existence, type et valeur

```
def test_variable():
    g = inspect.stack()[1].frame.f_globals
    assert "x" in g, "Variable x manquante."
    assert isinstance(g["x"], int), "x pas entier."
    assert g["x"] == 3, "mauvaise valeur."
```

Fonction

Définition : existence, arguments, docstring

```
def test_fonction():
    g = inspect.stack()[1].frame.f_globals
    assert "diff" in g, "Fonction diff manquante."

    sig = inspect.signature(g["diff"])
    assert len(sig.parameters)==2, "Nb arguments"
    assert list(sig.parameters.keys())==["a", "b"], \
        "Mauvais nom des arguments"
    assert sig.parameters["b"].default == 3,\
        "Valeur par défaut de b"
    assert g["diff"].__doc__ is not None, \
        "Docstring manquante."
```

Préconditions

Sous-entendu : une précondition non satisfaite lève une exception `AssertionError`, qui peut être interceptée avec `try…except`.

```
def test_div():
    g = inspect.stack()[1].frame.f_globals
    div = g["div"]
    try:
        div(4,0)
    except AssertionError:
        print("Précondition dénominateur non nul : OK.")
    except DivisionByZero:
        print("Précondition dénominateur non nul : NON.")
```

Valeurs de retour

```
def test_inc():
    g = inspect.stack()[1].frame.f_globals
    inc = g["inc"]
    assert inc(4) == 6, "Mauvaise valeur de retour."
```

Classes

Existence (classe, attributs, méthodes)

```
def test_point():
    g = inspect.stack()[1].frame.f_globals
    assert "Point" in g
    Point = g["Point"]
    p = Point(3)
    assert hasattr(p, "x"), "Attribut x manquant."
    assert hasattr(p, "move"), "Méthode move manquante."
    # ...
```

Autres tests

- tests sur les attributs (existence, type, valeur) => tests sur les variables
- tests sur les méthodes (existence, arguments, valeurs de retour) => tests sur les fonctions

Enjoliver

Messages

Assertions et messages d'erreur

- Moches
- Arrêt à la première erreur

Solution : `if`, ou `try…except…else` pour intercepter les exceptions et afficher des messages plus clairs.

Exemple I

```
def test_fonction():

    # Test existence
    g = inspect.stack()[1].frame.f_globals
    if "diff" not in g:
        print("[err] Fonction diff manquante.")
        return

    # Test définition
    sig = inspect.signature(g["diff"])
    if len(sig.parameters)!=2:
        print("[err] 2 arguments obligatoires.")
    if "a" not in sig.parameters:
        print("[err] Argument a manquant.")
```

Exemple II

```
if "b" not in sig.parameters:
    print("[err] Argument b manquant.")
else:
    print("[ok] Fonction bien définie.")

# Test préconditions
try:
    g["diff"](4,0)
except AssertionError:
    print("[err] Vérification dénominateur non nul.")
except DivisionByZero:
    print("[err] Vérification dénominateur non nul.")
else:
    print("[ok] Vérification dénominateur non nul.")
# ...
```

Couleurs, emojis, statistiques

Voir exemples de code réutilisable classe `Test`.

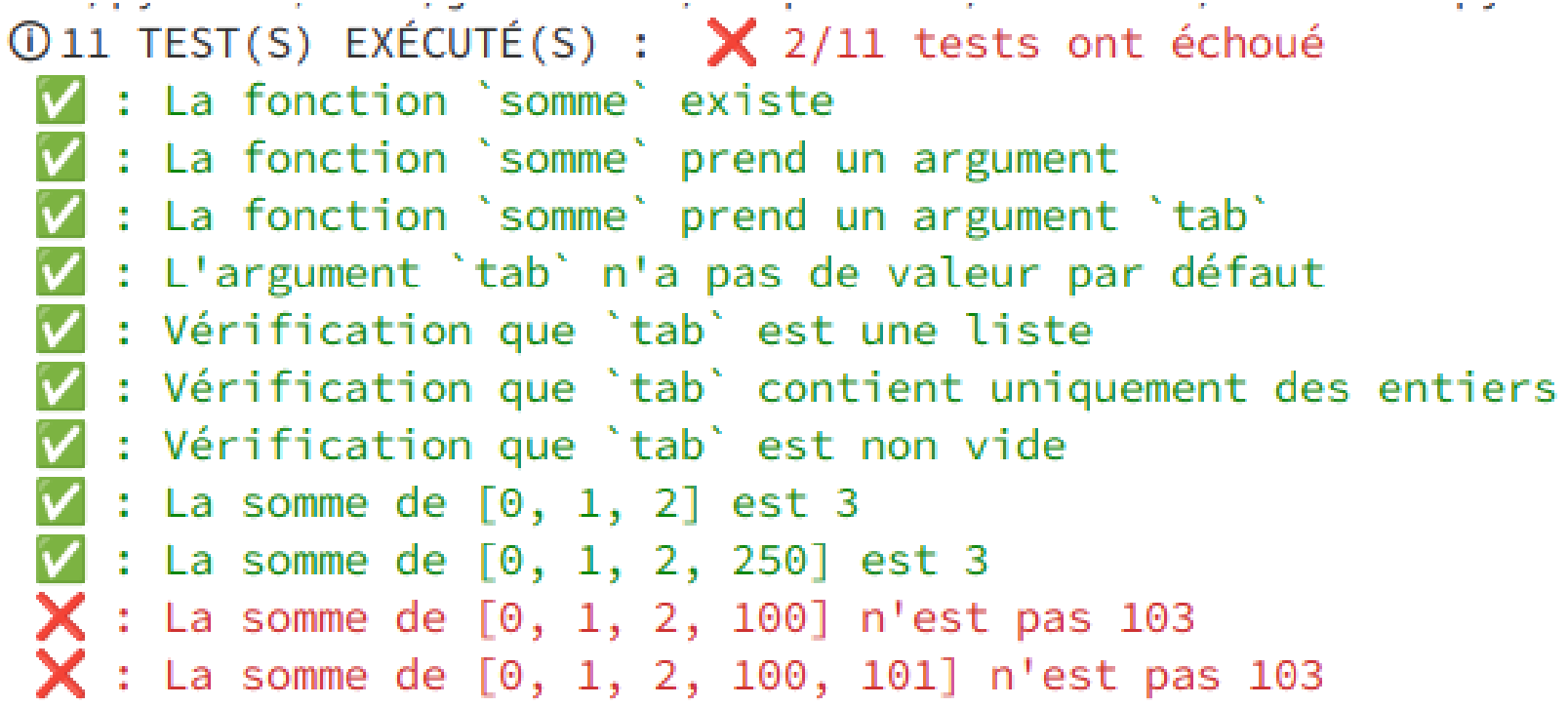


Figure 4: Exemple

Notebooks séquencés –> Manuel