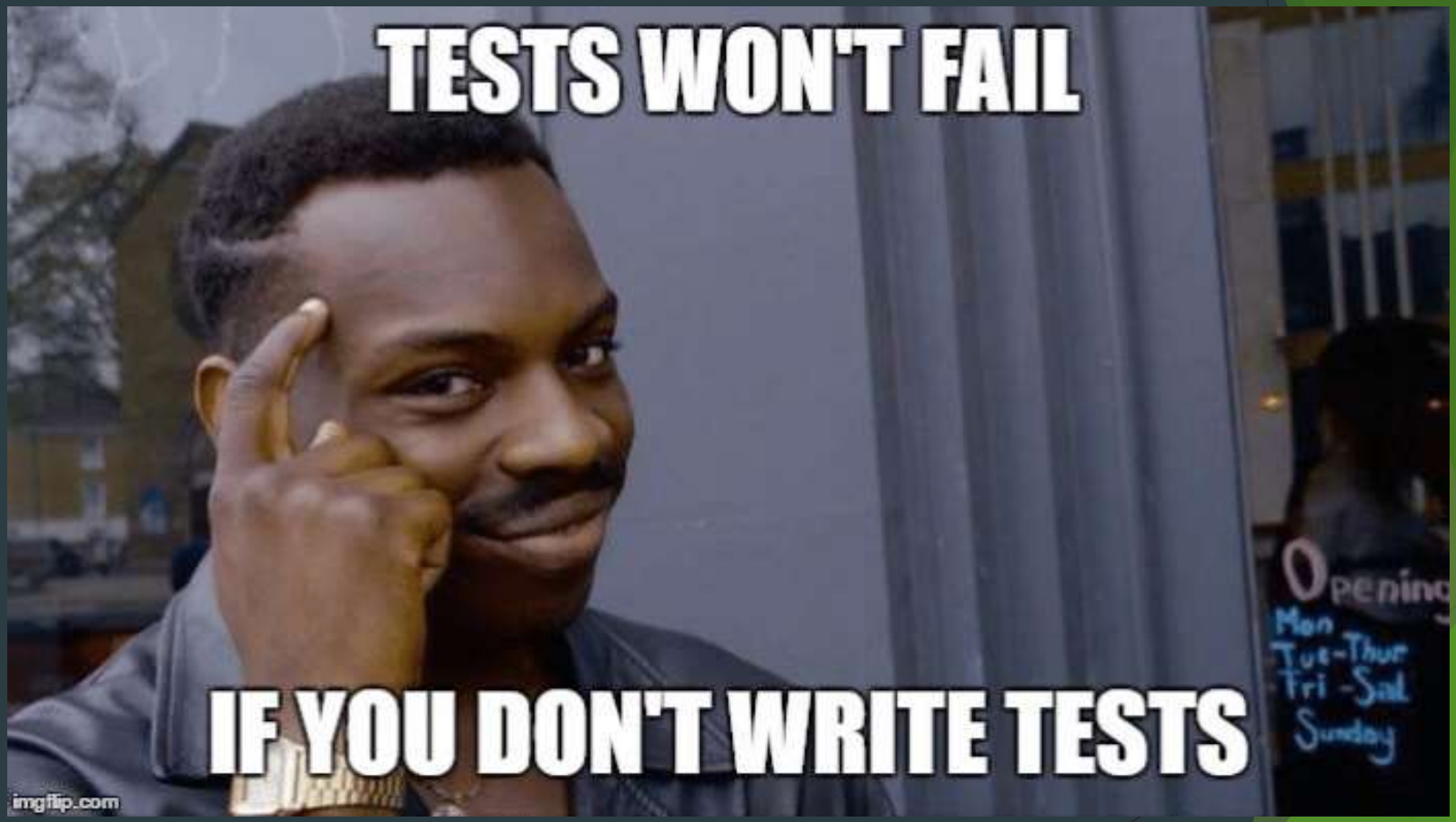
INFOB240 — Initiation à l'autoapprentissage par les MOOC

MOOC Software Testing – January 12th

By Yannis Van Achter

Road map

- Study case: Sudoku
- Software testing in my studies
- Real world software testing
- Question answer
  - Global software testing

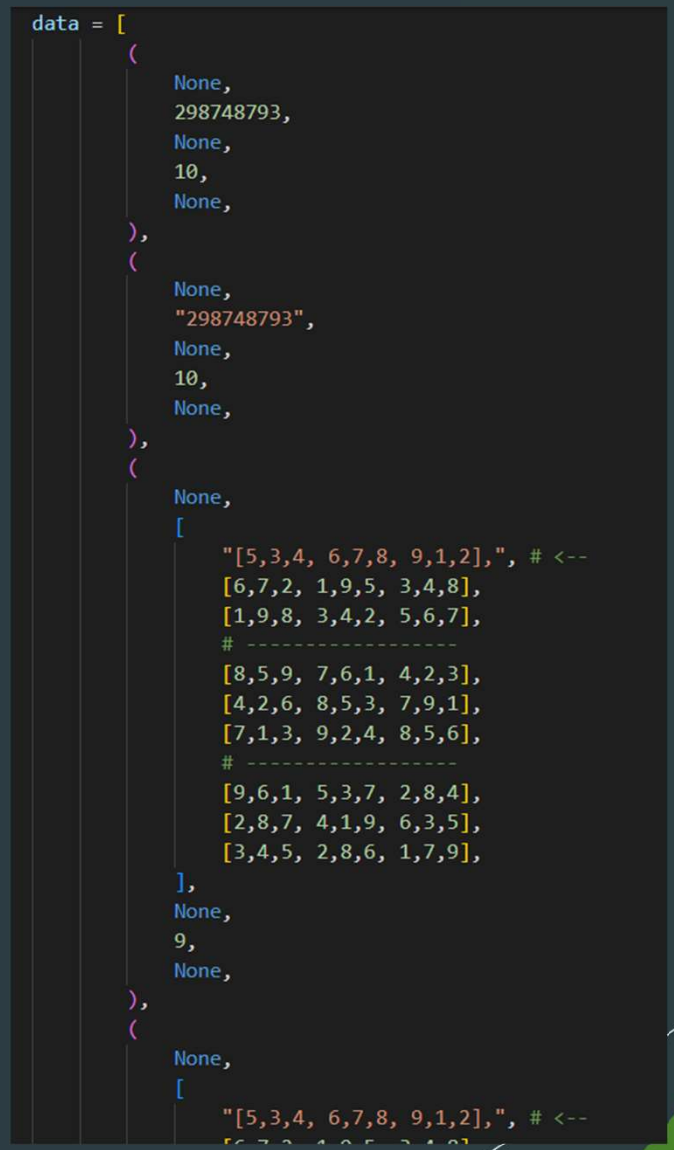

TESTS WON'T FAIL
IF YOU DON'T WRITE TESTS
imgflip.com
Opening
Mon
Tue-Thur
Fri-Sat
Sunday

Study case: Sudoku

Data set for my tests

- With txt file
  - On error exptectation: ast.literal_eval dit not understand it is an object (error object) and confused with sting
  - => Cancelled
- With JSON file
  - No Error object allowed
  - => Cancelled
- With Python file
  - All problem solved
  - BUT…

Can not write more test case in the list with save => Have to get input given and adding it manually to data file



```
data = [
        (
            None,
            298748793,
            None,
            10,
            None,
        ),
        (
            None,
            "298748793",
            None,
            10,
            None,
        ),
        (
            None,
            [
                "[5,3,4, 6,7,8, 9,1,2],", # <--
                [6,7,2, 1,9,5, 3,4,8],
                [1,9,8, 3,4,2, 5,6,7],
                # -----------------
                [8,5,9, 7,6,1, 4,2,3],
                [4,2,6, 8,5,3, 7,9,1],
                [7,1,3, 9,2,4, 8,5,6],
                # -----------------
                [9,6,1, 5,3,7, 2,8,4],
                [2,8,7, 4,1,9, 6,3,5],
                [3,4,5, 2,8,6, 1,7,9],
            ],
            None,
            9,
            None,
        ),
        (
            None,
            [
                "[5,3,4, 6,7,8, 9,1,2],", # <--
```

How did I build my data set ?

- Get cases gives by the MOOC
- Add invalid cases (int, str, float ...)
- Change some value type
- Giving a tuple in stead
- Change size

Errors during test writing

```
(venv) C:\Users\yanni\OneDrive\Documents\GitHub\INFOB240-Exam-work>c:/Users/yanni/OneDrive/Documents/GitHub/INFOB240-Exam-w
NFOB240-Exam-work/src/module/_test.py -v
test_check_sudoku (__main__.TestModule) ... ERROR
test_place_random_value (__main__.TestModule)
Note: Also test generate_grid() ... ok
test_solve_sudoku (__main__.TestModule) ... ok
test_try_solution (__main__.TestModule) ... ok

======================================================================
ERROR: test_check_sudoku (__main__.TestModule)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "c:\Users\yanni\OneDrive\Documents\GitHub\INFOB240-Exam-work\src\module\_test.py", line 84, in test_check_sudoku
    return_ = check_sudoku(grid, size)
  File "c:\Users\yanni\OneDrive\Documents\GitHub\INFOB240-Exam-work\src\module\checker.py", line 35, in check_sudoku
    raise TypeError("size must be an integer")
TypeError: size must be an integer

----------------------------------------------------------------------
Ran 4 tests in 301.740s

FAILED (errors=1)

(venv) C:\Users\yanni\OneDrive\Documents\GitHub\INFOB240-Exam-work>
```

Executing tests

```
(venv) C:\Users\yanni\OneDrive\Documents\GitHub\INFOB240-Exam-work>c:/Users/yanni/OneDriv
NFOB240-Exam-work/src/module/_test.py -v
test_check_sudoku (__main__.TestModule) ... ok
test_place_random_value (__main__.TestModule)
Note: Also test generate_grid() ... ok
test_solve_sudoku (__main__.TestModule) ... ok
test_try_solution (__main__.TestModule) ... ok

----------------------------------------------------------------------
Ran 4 tests in 382.623s

OK

(venv) C:\Users\yanni\OneDrive\Documents\GitHub\INFOB240-Exam-work>
```

+-6,3666 minutes

```python
# function make test easyer to read
def get_column(grid, column_id): ···

def get_sub_grid(grid, row_start, column_start, length=None): ···

def isUnique(value, under_test: list): ···

def get_start(size, current_start, length=None): ···

# https://docs.python.org/3.11/library/unittest.html
# https://www.youtube.com/watch?v=apgReCCAQr4
class TestModule(unittest.TestCase):
    def setUp(self): ···

    def test_check_sudoku(self): ···

    def test_solve_sudoku(self): ···

    def test_try_solution(self):  ···

    def test_place_random_value(self): ···
```

Well, tests succeed but are them well construct ?

```python
def test_place_random_value(self):
    """Note: Also test generate_grid() """
    # random testing (we start from valid input) to simulate user comportement
    perfect_square_number_list = [i * i for i in range(self.n_tests + 1)]
    seed_random_test = None  # random.seed  # HOWTO GET THE SEED ?
    with open("./TestPlaceRandomValue_return.txt", "w+") as file:
        file.write(f"Current seed: {seed_random_test}\n")
        for test_id in range(self.n_tests):
            size = random.randint(0, int(test_id**2.5))
            if size in perfect_square_number_list and size >= 4:
                empty_grid = generate_grid(size)
                discoverd = random.randint(1, size**2)
                if discoverd >= size**2 or discoverd < 0:
                    # https://stackoverflow.com/questions/61061723/python-unittest-unit-test-the-message-passed-in-raised-exception
                    with self.assertRaises(ValueError) as e:
                        place_random_value(empty_grid, size, discoverd)
                    exception = e.exception
                    self.assertEqual(
                        type(exception),
                        ValueError,
                        f"ValueError not raises with size of {size} and a n° of discoverd of {discoverd}",
                    )
                else:
                    randomly_placed_grid = place_random_value(
                        empty_grid, size, discoverd
                    )
                    count = 0
                    for row in randomly_placed_grid:
                        for column in row:
                            if column != 0:
                                count += 1
                    try:
                        assert (
                            count == discoverd
                        ), "Check function place random value placed {discoverd} but this is not the case.\n\t\tThe is only {count}"
                    except AssertionError as e:
                        file.write(
                            f"{e}\nExpected a count of: {discoverd}\nGet: {count}\n"
                            + ("=" * 20)
                            + "\n"
                        )
            else:
                # https://stackoverflow.com/questions/61061723/python-unittest-unit-test-the-message-passed-in-raised-exception
                with self.assertRaises(ValueError) as assert_raise:
                    generate_grid(size)
                exception_ = assert_raise.exception
                self.assertEqual(
                    type(exception_),
                    ValueError,
                    f"ValueError not raised by generate_grid\n\t\tAssert failed with {size}",
                )
```

```python
def test_check_sudoku(self):
    # full coverage testing
    for expected, grid, _, _, _ in self.Data:
        if expected in (ValueError, TypeError):
            # https://stackoverflow.com/questions/61061723/python-unittest-unit-test-the-message-p
            with self.assertRaises(expected) as e:
                check_sudoku(grid)
            self.assertEqual(
                type(e.exception), expected, f"{expected} not raised with grid of {grid}"
            )
        else:
            return_ = check_sudoku(grid)
            self.assertEqual(
                expected, return_, f"We expected {expected}\nNot {return_}"
            )

    for _, grid, _, size, expected in self.Data:
        if expected in (ValueError, TypeError):
            # https://stackoverflow.com/questions/61061723/python-unittest-unit-test-the-message-p
            with self.assertRaises(expected) as e:
                check_sudoku(grid, size)
            self.assertEqual(
                type(e.exception),
                expected,
                f"{expected} not raised with size of {size} and grid of {grid}",
            )
        else:
            return_ = check_sudoku(grid, size)
            self.assertEqual(
                expected, return_, f"We expected {expected}\nNot {return_}"
            )
```

So, we start from valid example, that we know the result

Ok if all is set by definition of test data, what does them do ?

- Case where we give and expect to have
  - Int -> None
  - Float -> None
  - Str -> None
  - Tuple -> None
  - Size of row change -> None
  - Two time the save value in a row or a column or a sub-grid -> False
  - Size given is not an int -> TypeError
  - Size given is not a perfect square -> ValueError
  - Size given < 4 -> ValueError

Software testing and my studies

INFOB131, INFOB132, INFOB231, INFOB233…

- Mistake in INFOB131 and INFOB132
  - Do not write test functions for projects
    - Lose use case of test I made only one time
  - Test by running and using application
    - Do not discover a lots of bugs
- Method developed in INFOB231 and INFOB233
  - Specification is one of the key to write efficient code that all team can use in project. The tests written from it is also more efficient and help to find more bugs early in project
  - When we created some data structure in INFOB231 we writhed first the test of our functions. And only when it's finished we writhed code
  - By creating a specification or an algorithm in INFOB233 we car write efficient test case and proof the functions we writes run correctly and follow the goal of the function

Real world testing

Real world testing: What we do

- Separate project in modules, class or function (named as work)
- Give one work to one person
- The person in charge of the work is responsible of his code (even if it fail)
- When we merge the work of everyone there is too much failure
- At implementation we don’t run each time the same test and add more if necessary (we change parameters but sometimes we don’t enter two time the same)

Real world testing: What we should do

- Separate project in modules, class or function (named as work)
- Give to one person the test to write for each work
  - Think about all use case and implement test for them
  - Check the eco-system of the project
    - Does everyone create code that do what it should ?
- Give to the other the writing of work
  - Run his code by the function of test written by his college
  - Update test written by his college (NEVER DELETE A TEST)

Test methods to combine

- Test coverage (by creator of the function)
- Random testing
  - By value (when failed value appear, we add the failed value in a file to improve test and check that program correctly respond in the future)
  - By path of execution (randomly increase or decrease an Queue, stack …)
- Create and run tests during development phase of the project and at each step
  - During we write a module
  - After writing the module
    - Check does each function does what it should do ?
  - After merge two module in the IS
    - Does the merge impact the quality of module ?
  - After building a back
  - After building a front
  - …

Question/Answer

Overall software testing

Method and operation of software testing

What is testing

- For given programs we will check
  - For the logic by comparing the expected output and effective output for specific input
  - The limits case (when we ask for number, give it as a string 😈)
  - Problematic case (give a sentence in stead of an int)
  - … (The limit of those test is your imagination)

- How to create testable programs ?
  - Clean code (write your code as a sentence (in measure of language you use))
  - Factoring
  - Modules
  - …

Unit testing

- Assertions
  - “assert” have to be True to continue the execution of the program
  - Do not use to check parameters in functions (instead raise exceptions)
  - Do not use to check that 1 +1 == 2, if this is false this is Python which is break not your code => Efficient test
  - Why using it ?
    - Auto test code
    - Stop execution before an bug arrive and allow you to know what’s going wrong during development phase
    - Pres-conditions and post-condition directly in the code (INFOB233)
  - Syntax: assert condition, “Commentary to help for debug”
- Unittest module of python
  - Useful for complex test (like is there an error which is raised at execution the test for the given arguments)
  - Give us more explanation about test running and exceptions that occurred (like time of execution for all tests)

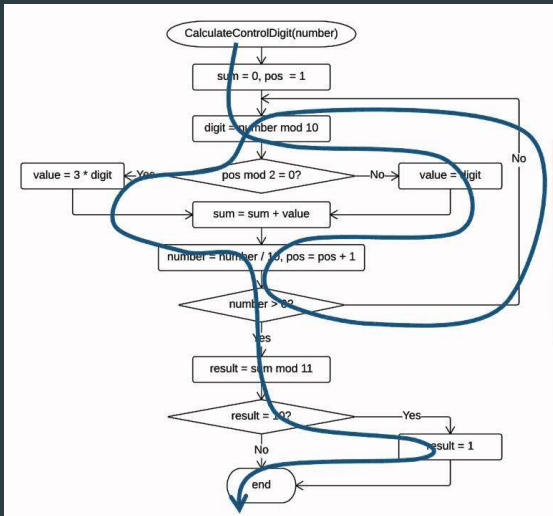
```
def generate_random_sequence(len_sequence: int) -> (str):
    """Genarate DNA sequence

    Parameter:
    ----------
        len_sequence (int): size we want for the DNA sequence

    Raises:
    -------
        TypeError: If len_sequence is not an integer
        ValueError: If len_sequence is lower or equal to 0

    Return:
    -------
        final_sequence (str): Final sequence wich respectet the algorithm of Grytezuk
    """
```

Type of testing: Black box

```python
def generate_random_sequence(len_sequence: int) -> (str):
    """Genarate DNA sequence···
    if not isinstance(len_sequence, int):
        raise TypeError("len_sequence must be an integer")
    if not len_sequence > 0:
        raise ValueError("len_sequence must be a positive number")

    # init loops variable
    final_sequence = ""
    nucleotides = ('A', 'C', 'T', 'G')

    # init loop
    while len(final_sequence) < len_sequence:
        final_sequence += rd.choice(nucleotides)
        print(f"Current sequence: {final_sequence}")
        is_contain, len_delete = contain_sub_sequence(final_sequence)
        if is_contain:
            final_sequence = final_sequence[:-len_delete] # get sub list from index 0 to last wanted to ke

    return final_sequence
```

Type of testing: White Box

Coverage testing: test each line of a code



Testing strategy

- Path coverage
- Branch coverage

Unit Tests – Path Coverage

Input	Output
x	x
3	3
31	1

- Enter in all bloc after a condition
- Run to execute all lines
- There is tools to help you to see which line wasn't executed
- Use with Whitebox Testing (we need to see the condition which are apply)

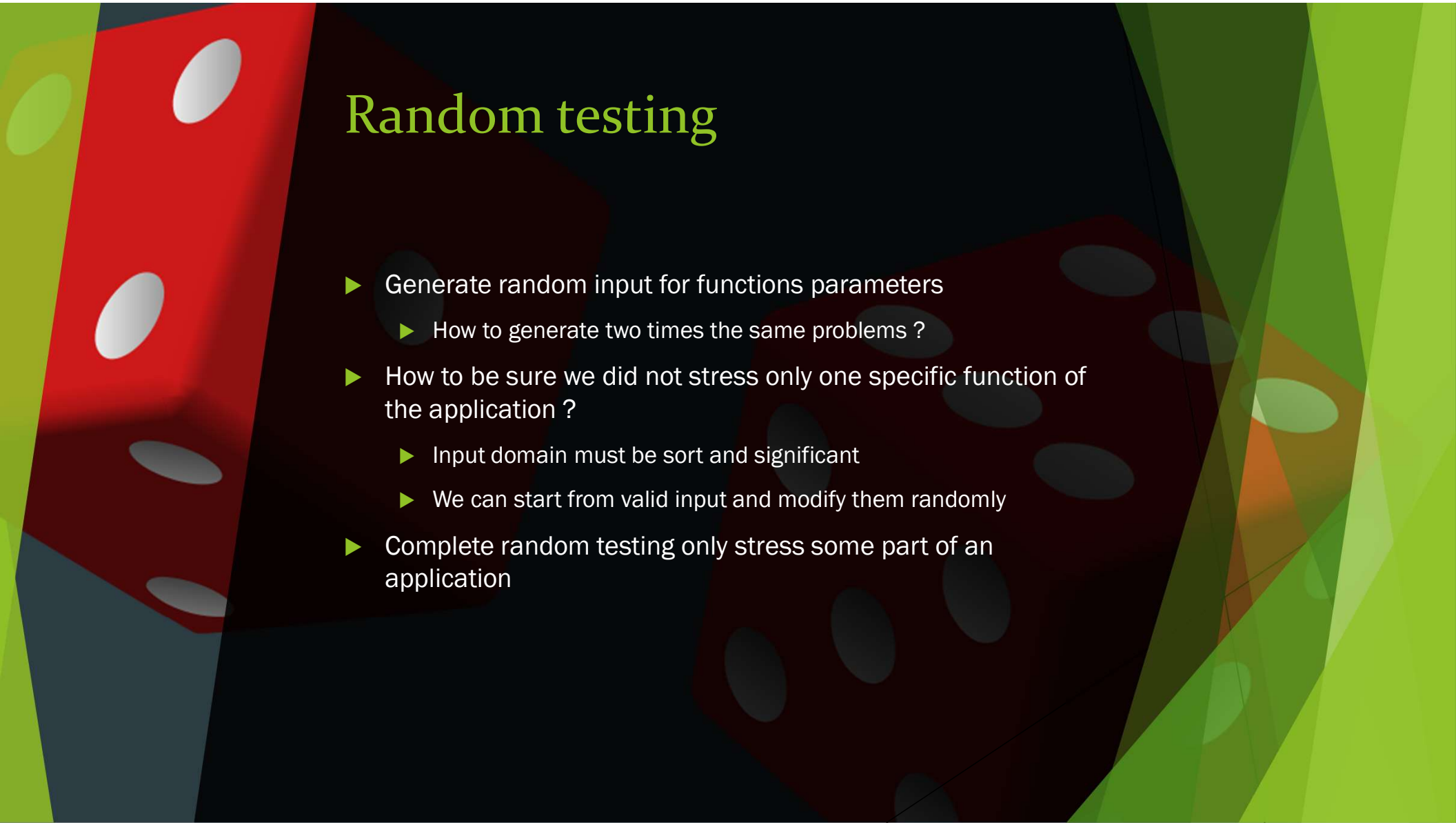
Coverage testing: test each line of a code

+

- Test all written code
- Avoid cognitive bias when two different people write test and SUT

-

- Big requirement in analysis before to write efficient test
- Cognitive bias when the person who write test, wrote/write the SUT



Random testing

- Generate random input for functions parameters
  - How to generate two times the same problems ?
- How to be sure we did not stress only one specific function of the application ?
  - Input domain must be sort and significant
  - We can start from valid input and modify them randomly
- Complete random testing only stress some part of an application

Random testing

+

- So we create software to do random testing
- Increase the test quality -> less human cognitive bias
- Less human intervention once it is automatised -> easy to maintain
- Surprise us

-

- Create valid input is difficult
- Test infinitively (no criteria to stop testing)
- Can find frequently the same bug
- On update of a fuzzer it can find a lots of other bugs (depression be like 😔)

Other methods of test

- Implementation testing
  - Test two modules together
- Differential testing
  - Compare two implementation
    - Use when we change an algorithm to check he still return or doing the same thing
- System testing
  - Does the system make the job ?
- Stress testing (pure random testing or data wave on a network for instance)
  - In network, does an API respond when there is a lot of different request ?
  - Does the execution/API call impact the quality of the function ?
- ...

Software testing is like being the hero and the antagonist in the same story