

# Introduction to Data Science

MODULE I – PART I

REPRODUCIBILITY IN PYTHON

Prof Sergio Serra e Jorge Zavaleta

# Reproducible Experiment in CS

---

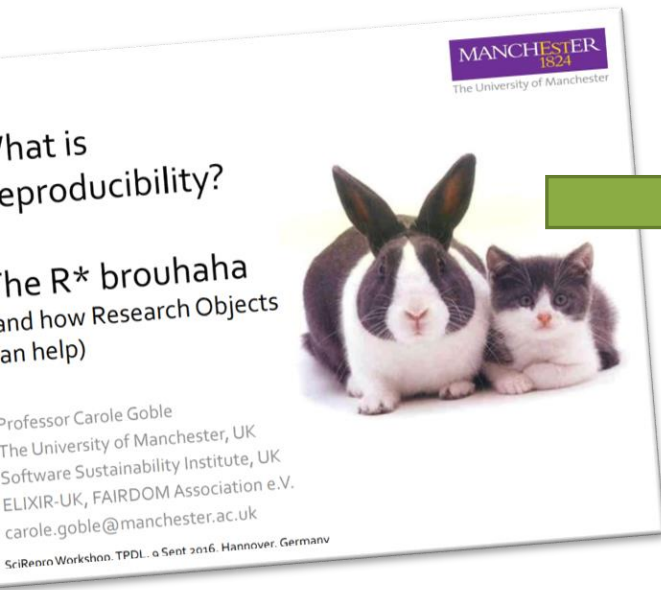
“An experiment composed by a sequence of steps  $S$  that has been developed at time  $T$ , on environment (hardware and OS)  $E$ , and on data  $D$  **is reproducible if** it can be executed with a sequence of steps  $S'$  (different or the same as  $S$ ) at time  $T' > T$ , on environment  $E'$  (different or the same as  $E$ ), and on data  $D'$  (different or the same as  $D$ ) with **consistent results** ( $R$  and  $R'$  consistent)”.

This definition includes both **exact reproducibility (repeatability)** and **approximate reproducibility**

- **Exact Reproducibility** requires reproducing the exact same result
  - $S' = S$  and  $E' = E$  and  $D' = D \Rightarrow R = R'$
- **Approximate Reproducibility** involves producing similar results as the original ones
  - $S' \neq S$  or  $E' \neq E$  or  $D' \neq D \Rightarrow R \sim R'$



# 1P5R



R1

*rerun*

Robust

*variations on  
experiment and set up*

R2

*repeat*

Defend

*same experiment,  
same set up, same lab*

Validate

R3

*reproduce*

Compare

*variations on experiment,  
on set up, independent labs*

R4

*reuse*

Transfer

*different experiment*

R5

*replicate*

Certify

*same experiment,  
same set up, independent lab*

For a program to  
contribute to science,  
it should be rerunnable  
(R1), repeatable (R2),  
reproducible (R3),  
reusable (R4),  
replicable (R5)



R0

Productivity

Track differences

# R0 – Random Walk

Environment info is unknown. Does it work on any Python version?

In [23]: ▶ # Random walk (R0: worst you can do)

```
import random

x = 0
for i in xrange(10):
    step = random.choice([-1,+1])
    x += step
    print x,
```

**NameError** Traceback (most recent call last)  
<ipython-input-23-ff098a3b5866> in <module>  
3  
4 x = 0  
----> 5 for i in xrange(10):  
6 step = random.choice([-1,+1])  
7 x += step  
  
**NameError**: name 'xrange' is not defined

Python 2

In [21]: ▶ # Random walk (R0: worst you can do)

```
import random

x = 0
for i in range(10):
    step = random.choice([-1,+1])
    x += step
    print (x),
```

-1  
0  
1  
2  
1  
2  
3  
2  
1  
0

Python 3

# R1 – Re-runnable

Environment Info. Scientist is responsible for keeping this info!

```
In [25]: ▶ # Random walk (R1: re-runnable)
          # Tested with Python 3
          import random

          x = 0
          walk = []
          for i in range(10):
              step = random.choice([-1,+1])
              x += step
              walk.append(x)

          print(walk)

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

**Re-runnable code** should describe—with enough details to be recreated—an execution environment in which it is executable.  
**It is far from being either obvious or easy.**



# R2 - Repeatable

```
In [30]: ▶ # Random walk (R2: repeatable)
# Tested with Python 3
import random

random.seed(1) # RNG initialization

x = 0
walk = []
for i in range(10):
    step = random.choice([-1,+1])
    x += step
    walk.append(x)

print(walk)
# Saving output to disk
with open('results-R2.txt', 'w') as fd:
    fd.write(str(walk))
```

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

A repeatable code is one that can be rerun and that produces the same result on successive runs

- Program needs to be deterministic
- Control the initialization of pseudo-random number generators

Previous results need to be available (it is possible to compare with current results)

- $S' = S$  and  $E' \sim E$  and  $D' = D$  and  $R = R'$

# R2 - Repeatable

initialization of Random Seed!

```
In [30]: ▶ # Random walk (R2: repeatable)
# Tested with Python 3
import random

random.seed(1) # RNG initialization

x = 0
walk = []
for i in range(10):
    step = random.choice([-1,+1])
    x += step
    walk.append(x)

print(walk)
# Saving output to disk
with open('results-R2.txt', 'w') as fd:
    fd.write(str(walk))
```

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

- Verifying the qualitative aspects of the results and the conclusions that are made are not tied to a specific initialization of the pseudo-random generator is an integral part of any scientific undertaking in computational Science
- This is usually done by repeating the simulations multiple times with different seeds

Due to a **change** that occurred in the **pseudo-random number generator** between Python 3.2 and Python 3.3, executing this code in Python 3.3 **WILL NOT** generate the same results when compared to the Python 3.2 execution.

# R2 - Repeatable

Repeatable Random Walk  
Example is not reproducible!

```
In [30]: ▶ # Random walk (R2: repeatable)
# Tested with Python 3
import random

random.seed(1) # RNG initialization

x = 0
walk = []
for i in range(10):
    step = random.choice([-1,+1])
    x += step
    walk.append(x)

print(walk)
# Saving output to disk
with open('results-R2.txt', 'w') as fd:
    fd.write(str(walk))
```

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

- Executed with Python 2.7–3.2, the code will produce the sequence

-1, 0, 1, 0, -1, -2, -1, 0, -1, -2

- But with Python 3.3–3.6, it will produce

-1, -2, -1, -2, -1, 0, 1, 2, 1, 0

- With future versions of the language, it may change still



# R2 - Repeatable

Repeatable Random Walk  
Example is not reproducible!

```
In [30]: ▶ # Random walk (R2: repeatable)
# Tested with Python 3
import random

random.seed(1) # RNG initialization

x = 0
walk = []
for i in range(10):
    step = random.choice([-1,+1])
    x += step
    walk.append(x)

print(walk)
# Saving output to disk
with open('results-R2.txt', 'w') as fd:
    fd.write(str(walk))
```

Save output to allow comparing different runs

[-1, -2, -1, -2, -1, 0, 1, 2, 1, 0]

# R3 - Reproducible

Use notepad to keep track of code versions

Test for reproducibility

Record environment  
with output data

```
In [43]: # Copyright (c) 2017 N.P. Rougier and F.C.Y. Benureau
# Adapted by Serra
# Release under the Windows 10
# Tested with 64 bit (AMD64)
import sys, subprocess, datetime, random

def compute_walk():
    x = 0
    walk = []
    for i in range(10):
        if random.uniform(-1, +1) > 0:
            x += 1
        else:
            x -= 1
        walk.append(x)
    return walk

# If repository is dirty, don't run anything
if subprocess.call(("notepad", "diff-index",
                    "--quiet", "HEAD")):
    print("Repository is dirty, please commit first")
    sys.exit(1)

# Get git hash if any
hash_cmd = ("notepad", "rev-parse", "HEAD")
revision = subprocess.check_output(hash_cmd)

# Unit test
random.seed(42)
assert compute_walk() == [1,0,-1,-2,-1,0,1,0,-1,-2]

# Random walk for 10 steps
seed = 1
random.seed(seed)
walk = compute_walk()

# Display & save results
print(walk)
results = {
    "data"      : walk,
    "seed"      : seed,
    "timestamp" : str(datetime.datetime.utcnow()),
    "revision"  : revision,
    "system"    : sys.version}
with open("results-R3.txt", "w") as fd:
    fd.write(str(results))
```

[-1, 0, 1, 0, -1, -2, -1, 0, -1, -2]

# R4 - Reusability

```
In [45]: ▶ import sys, subprocess, datetime, random

def compute_walk(count, x0=0, step=1, seed=0):
    """Random walk
    count: number of steps
    x0 : initial position (default 0)
    step : step size (default 1)
    seed : seed for the initialization of the
    random generator (default 0)
    """
    random.seed(seed)
    x = x0
    walk = []
    for i in range(count):
        if random.uniform(-1, +1) > 0:
            x += 1
        else:
            x -= 1
        walk.append(x)
    return walk

def compute_results(count, x0=0, step=1, seed=0):
    """Compute a walk and return it with context"""
    # If repository is dirty, don't do anything
    if subprocess.call(("notepad", "diff-index",
                       "--quiet", "HEAD")):
        print("Repository is dirty, please commit")
        sys.exit(1)
```

```
# Get git hash if any
hash_cmd = ("notepad", "rev-parse", "HEAD")
revision = subprocess.check_output(hash_cmd)

# Compute results
walk = compute_walk(count=count, x0=x0,
                    step=step, seed=seed)

return {
    "data" : walk,
    "parameters": {"count": count, "x0": x0,
                  "step": step, "seed": seed},
    "timestamp" : str(datetime.datetime.utcnow()),
    "revision" : revision,
    "system" : sys.version}

if __name__ == "__main__":
    # Unit test checking reproducibility
    # (will fail with Python<=3.2)
    assert (compute_walk(10, 0, 1, 42) ==
            [1,0,-1,-2,-1,0,1,0,-1,-2])

    # Simulation parameters
    count, x0, seed = 10, 0, 1
    results = compute_results(count, x0=x0, seed=seed)

    # Save & display results
    with open("results-R4.txt", "w") as fd:
        fd.write(str(results))
    print(results["data"])
```

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

# R4 – Reusability

---

Making your program **reusable** means it can be **easily used**, and **modified**, by you and other people, **inside and outside your lab**

The easier it is to use your code, the lower the threshold is for other to study, modify and extend it

- This implies it should be **well documented**

Scientists constantly face the constraint of time

- if a model is **available, documented**, and can be installed, run, and **understood** all in a few hours, it will be preferred over another that would require weeks to reach the same stage

A **reproducible** and **reusable** code offers a platform both verifiable and easy-to-use, fostering the development of derivative works by other researchers on solid foundations •

Those derivative works **contribute to the impact of your original contribution (citations!!)**

# R4 – Reusability – Tips

---

## Avoid hardcoded or magic numbers

- **Magic numbers** are those present directly in the source code (no name, no semantics)
- **Hardcoded values** are variables that cannot be changed through an argument or a parameter configuration file
  - Example: R3 Random Walk, the seed is hardcoded, and the number of steps is a magic number

## Code behavior should not be changed by commenting/uncommenting code

- Instead, it should be **explicitly set through parameters** that are accessible to the end user
- This improves reproducibility in two ways
  - it allows those conditions to be recorded as parameters in the result files, and
  - it allows to define separate scripts to run or configuration files to load to produce each of the figures of the published paper

# R5 – Replicability

---

Replicating implies **writing a new code** matching the conceptual description of the article, in order to obtain the same (**compatible**) results

- $S' \neq S$  and  $(E' \neq E \text{ or } D' \neq D) \Rightarrow \mathbf{R \sim R'}$

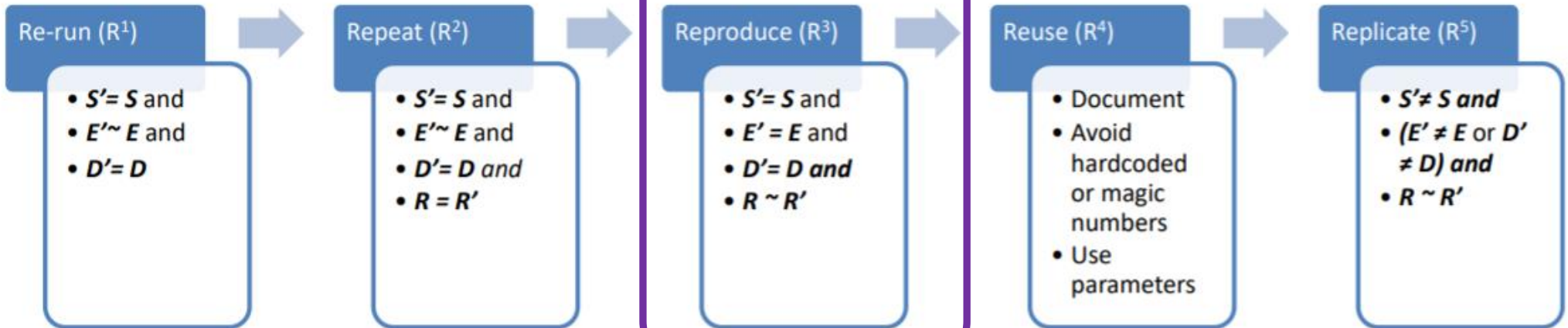
Replication affords **robustness** to the results

- if the original code contain an error, a different codebase creates the possibility that this error will not be repeated
- Every paper is a mistake if a parameter is forgotten
- Replication efforts use the paper first, and then the reproducible code that comes along with it whenever the paper falls short of being precise enough



# Summary

## Minimum Scientific Standard



Code (local) + Environment + Input Data

Same (Compatible) Output

Same Environment

Publicly available +  
Documentation



# No Jupyter

## PPGI/UFRJ 2020.3

Prof Sergio Serra e Jorge Zavaleta

## Reprodutibilidade em Python

Fonte: Re-run, Repeat, Reproduce, Reuse, Replicate: Transforming Code into Scientific Contributions Fabien C. Y. Benureau and Nicolas P. Rougier

"Replicability is a cornerstone of science. If an experimental result cannot be re-obtained by an independent party, it merely becomes, at best, an observation that may inspire future research (Mesirov, 2010; Open Science Collaboration, 2015)."

## R0 - Irreproducibility

A program can fail as a scientific contribution in many different ways for many different reasons, i.e. code errors; deprecated methods; older compiler versions, lack of documentation, ...

```
In [ ]: ▶ import random

        for i in xrange(10):                #xrange?
            step = random.choice([-1,+1])
            x += step
        print x,                             #print?
```

## R1 - Re-Runnable

Re-runnable code should describe—with enough details to be recreated—an execution environment in which it is executable. It is far from being either obvious or easy.

```
In [ ]: ▶ # Random walk (R1: re-runnable)
        # Tested with Python 3.8
```

# References

---

Benureau, F., Rougier, N. Re-run, Repeat, Reproduce, Reuse, Replicate: Transforming Code into Scientific Contributions. *Frontiers in Neuroinformatics*. V.11, article 69, 2018.

Freire, J.; Chirigati, F. Provenance and the Different Flavors of Computational Reproducibility. *IEEE Data Engineering Bulletin*. V. 41:15-26, 2018.

Goble, C. What is reproducibility? The Rbrouhaha, In:First International Workshop on Reproducible Open Science (Hannover), 2016.

Goble, C. <http://repscience2016.research-infrastructures.eu/img/CaroleGoble-ReproScience2016v2.pdf>