Python Setup in R studio and some Basic

Python vs R

What is the main similarity between python and R?

Both open source programming languages are supported by large communities, continuously extending their libraries and tools.

What is the main difference between python and R?

- Python is a general-purpose, object-oriented programming language. Released in 1989. It is prefer among deep and machine learning researcher.
- R is optimized for statistical analysis and data visualization. Developed in 1992. Popular among data science scholars and researchers.
- In summary: Python is a flexible programming language used for various tasks like web development, data handling, and machine learning. On the other hand, R is specifically designed for statistical programming. Python performs better than R for tasks that aren't related to statistics, while R outperforms Python when it comes to statistical analysis.

Python

The {reticulate} package includes a Python engine for R Markdown that enables easy interoperability between Python and R chunks.

- Python Chunks Python code chunks work exactly like R code chunks: Python code is executed and any print or graphical output is included within the document.
- Most Python pros use Jupyter Notebook as their IDE when developing in Python. You may use online platforms like Google Colab or onine-python.com or download PyCharm.

- But for R programmers can use R Studio and the {reticulate} package.
  - Install and load {reticulate} package

R

```
install.packages("reticulate")
```

R

```
library(reticulate)
```

- To install a version of Python that reticulate will use,

R

```
reticulate::install_miniconda()
```

- To validate that Python was installed and is available, run

R

```
reticulate::py_available()
```

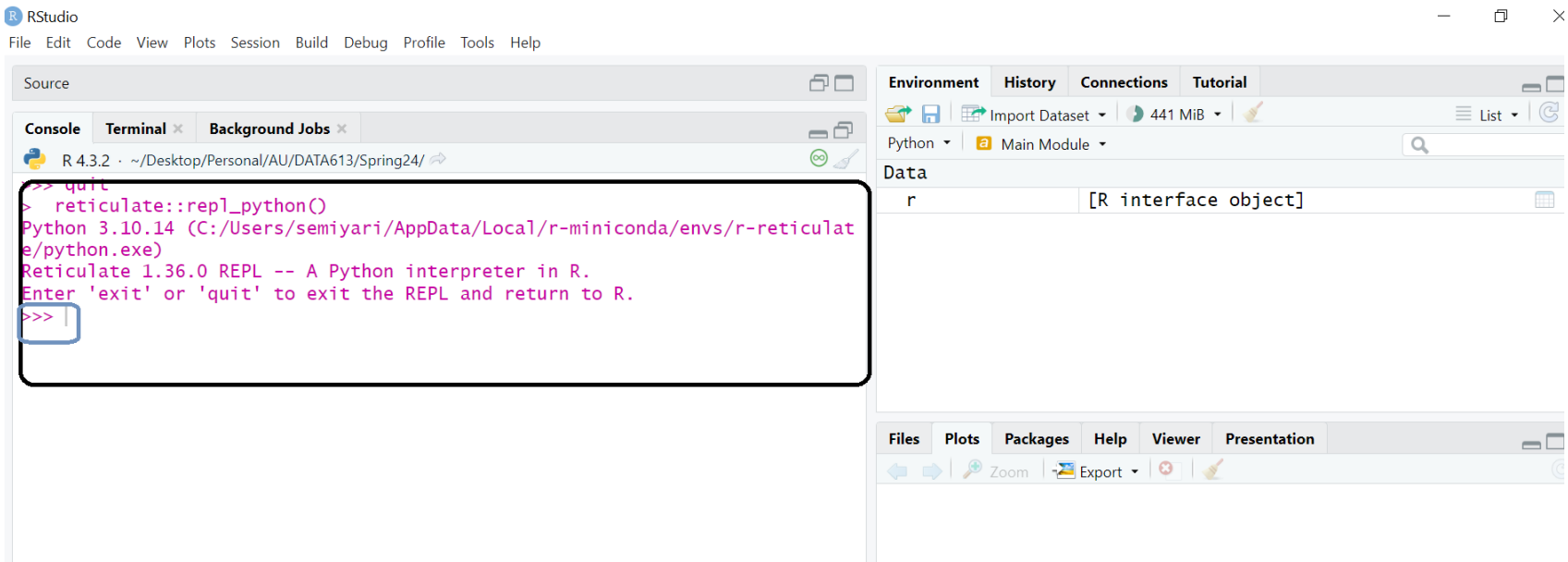
- To install Python packages, use `py_install()`. For example, we can install the `numpy`, `pandas`, and `matplotlib` packages via R

```
reticulate::py_install(
  packages = c("numpy", "pandas", "matplotlib")
  )
```

To start an IPython shell — similar to the R command promp — run the following in R:

R

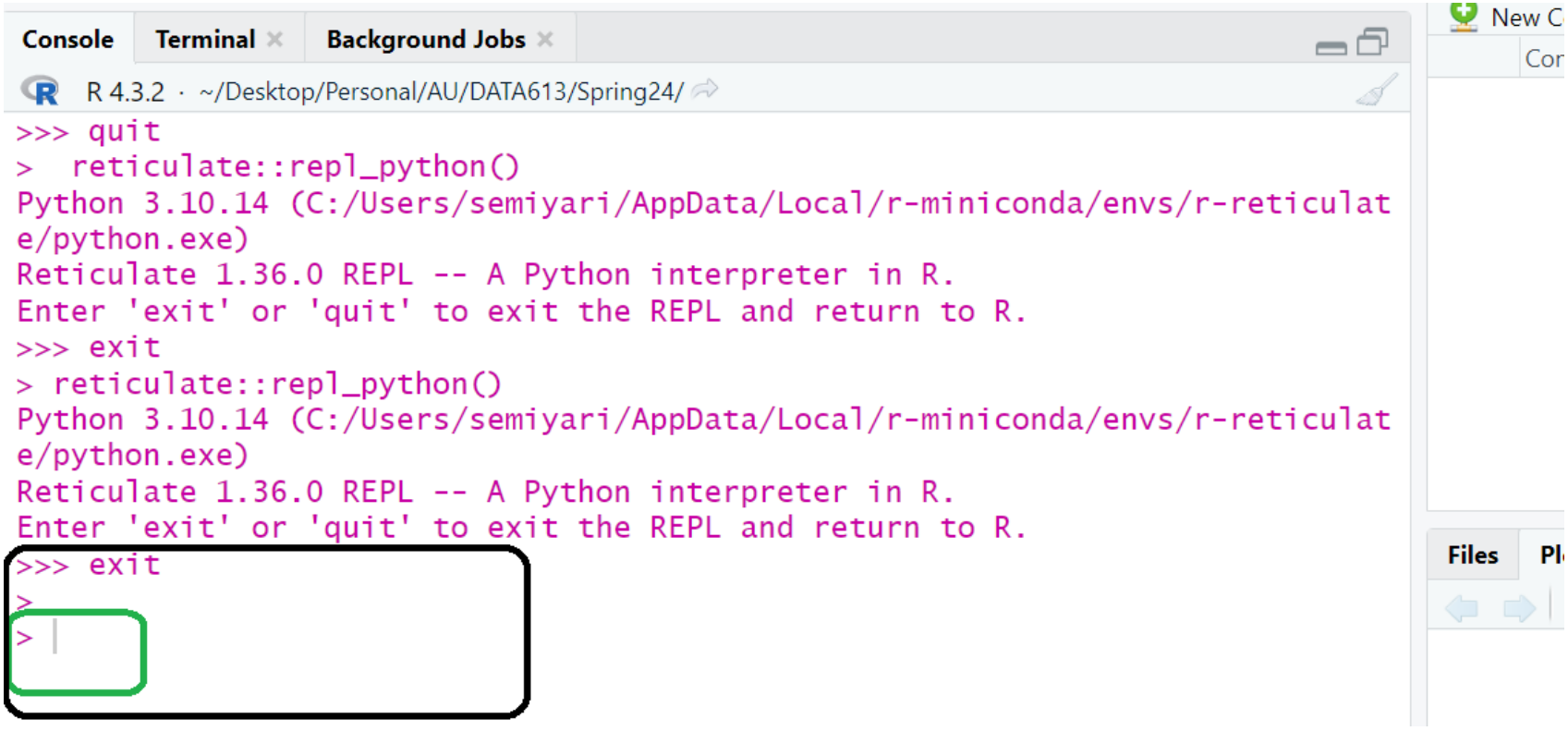
```
reticulate::repl_python()
```



- You can exit the REPL by typing the following:

Python

```
exit
```



- Objects are divided into two main parts

1. SCALAR
2. NON_SCALAR

1. SCALAR: Can not be subdivided, Example: int, float, bool, Nonetype!

- The NoneType object is a special type in Python and has only one value `None`. It represents the absence of a value. It is used to indicate that a variable or expression does not have a value or has an undefined value.

Python

```
print(print())
```

```
None
```

Python

```
x, y = None, 3
print("x = ",x, " and y =", y)
```

```
x =  None  and y = 3
```

- Type of an object, use `type()`

Python

```
print(type(y))
```

```
<class 'int'>
```

```
print(type(x))
```

```
<class 'NoneType'>
```

For the following example we use the package {`math`}. In Python we need to type `package.object`. for instance to use `pi` we need to write `math.pi`.

Python

```
import math
print(math.pi)
```

```
3.141592653589793
```

```
print(math.e)
```

```
2.718281828459045
```

```
print(math.sqrt(2))
```

```
1.4142135623730951
```

Python

```
print(type(math.pi))
```

```
<class 'float'>
```

```
print(type(math.e))
```

```
<class 'float'>
```

```
print(type(math.sqrt(2)))
```

```
<class 'float'>
```

- Type conversion (CAST): Casting is a process of converting a variable to another data type (object)

Python

```
x1 = float(15)
print("Type of ", 15, "is: ", type(15),
"\n and type of float(15) is: ", type(x1),
"\n and its value is: ", x1)
```

```
Type of  15 is:  <class 'int'>
 and type of float(15) is:  <class 'float'>
 and its value is:  15.0
```

Python

```
x2 = int(15.9)
print("Type of ", 15.9, "is: ", type(15.9),
"\n and type of int(15.9) is: ", type(x2),
"\n and its value is: ", x2)
```

```
Type of  15.9 is:  <class 'float'>
 and type of int(15.9) is:  <class 'int'>
 and its value is:  15
```

Python

```
Condition = True
print("Type of  the object Condition is \n",
type(Condition),
" \n and the object Condition is equal to ",
Condition)
```

```
Type of  the object Condition is
 <class 'bool'>
 and the object Condition is equal to  True
```

2. NON-SCALAR: Have internal structure. Example, strings, lists and tuples

- STRINGS: It is a data structure that represents a sequence of character.

Python

```
string_1 = "This is a string"
string_2 = "I think strings are funny"
```

Python

```
print(string_1 + string_2)
```

```
This is a stringI think strings are funny
```

Python

```
print("I have ", 2, "strings. The first one is: ",
string_1, "\nThe second one is", string_2)
```

```
I have  2 strings. The first one is:  This is a string
The second one is I think strings are funny
```

COMPARISON OPERATORS (INT, FLOAT, STRING)

Python

```
print(math.pi == 22/7)
```

```
False
```

```
print(math.pi > math.e)
```

```
True
```

Python

```
print(6 == math.factorial(3))
```

```
True
```

```
print(math.sin(math.pi/2) <= 1.0)
```

```
True
```

```
print(math.sin(math.pi/2) >= 1.0)
```

```
True
```

Python

```
print(7%2 != 1)
```

```
False
```

Python

```
string_1 = "This is a string"
string_2 = "I think strings are funny"
```

Python

```
print("The string_1 is: ", string_1,
"\nThe string_1 is:", string_2)
```

```
The string_1 is:  This is a string
The string_1 is: I think strings are funny
```

```
print(string_1 == string_2)
```

```
False
```

Python

```
print(string_1 == "This is a string")
```

```
True
```

```
print(string_1 >= "This is a string")
```

```
True
```

```
print(string_1 <
"This is a string and it is bigger than the other one")
```

```
True
```

LOGIC OPERATORS (BOOL)

1. NOT p
2. p and q
3. p or q

p	q	NOT p	p and q	p or q
T	T	F	T	T
T	F	F	F	T
F	T	T	F	T
F	F	T	F	F

Access R and Python objects:

- You can access R objects in Python
- You can access Python objects in R

- You can access R objects in Python using the `r` object. That is, `r.x` will access, in Python, the `x` variable defined using R.

R

```
x <- c(1, 4, 6, 2)
```

Python

```
r.x
```

```
[1.0, 4.0, 6.0, 2.0]
```

- You can access Python objects in R using the `py` object. That is, `py$x` will access, in R, the `x` variable defined using Pythong.

  Python

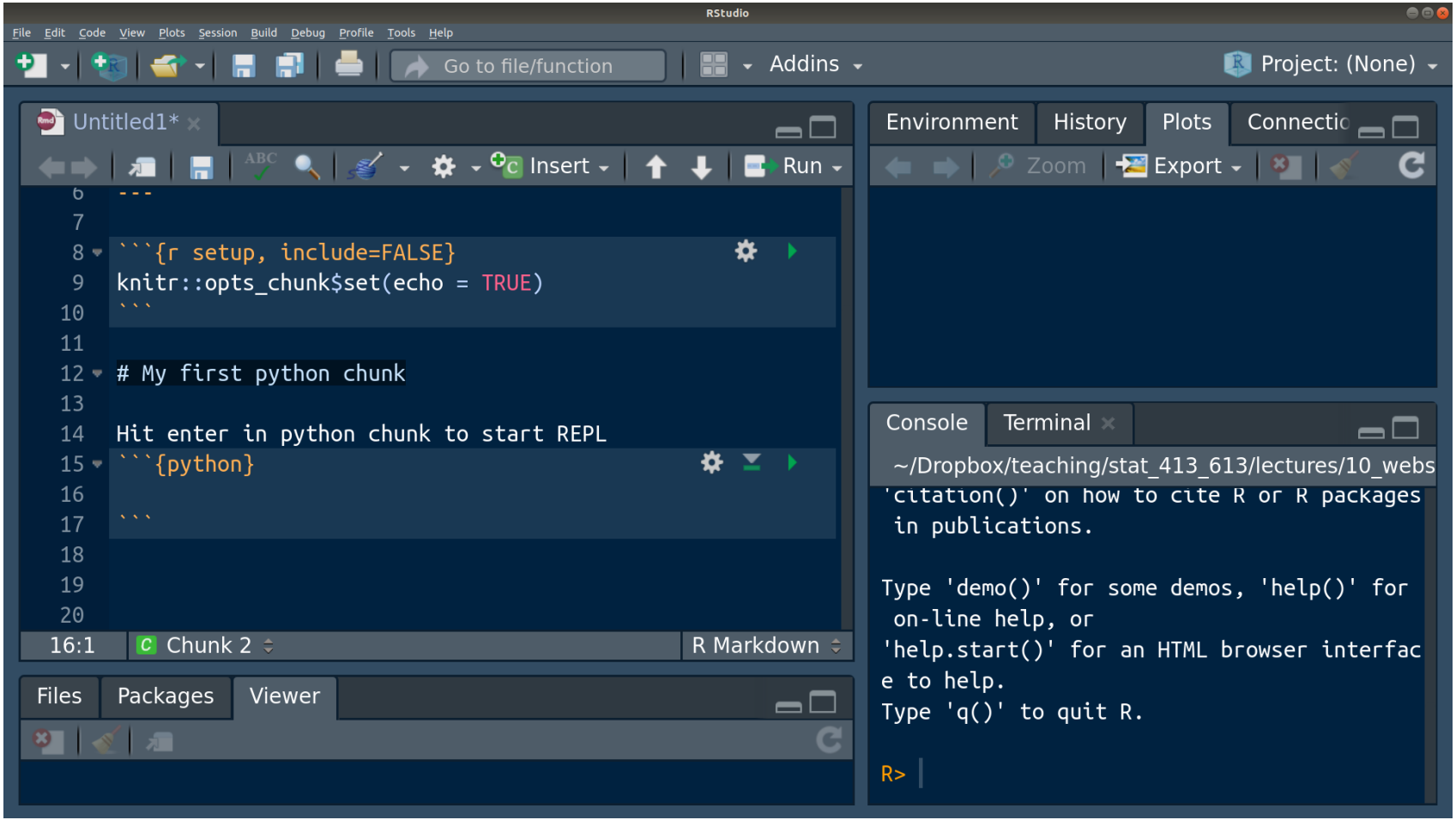
```
x = [8, 9, 11, 3]
```

R

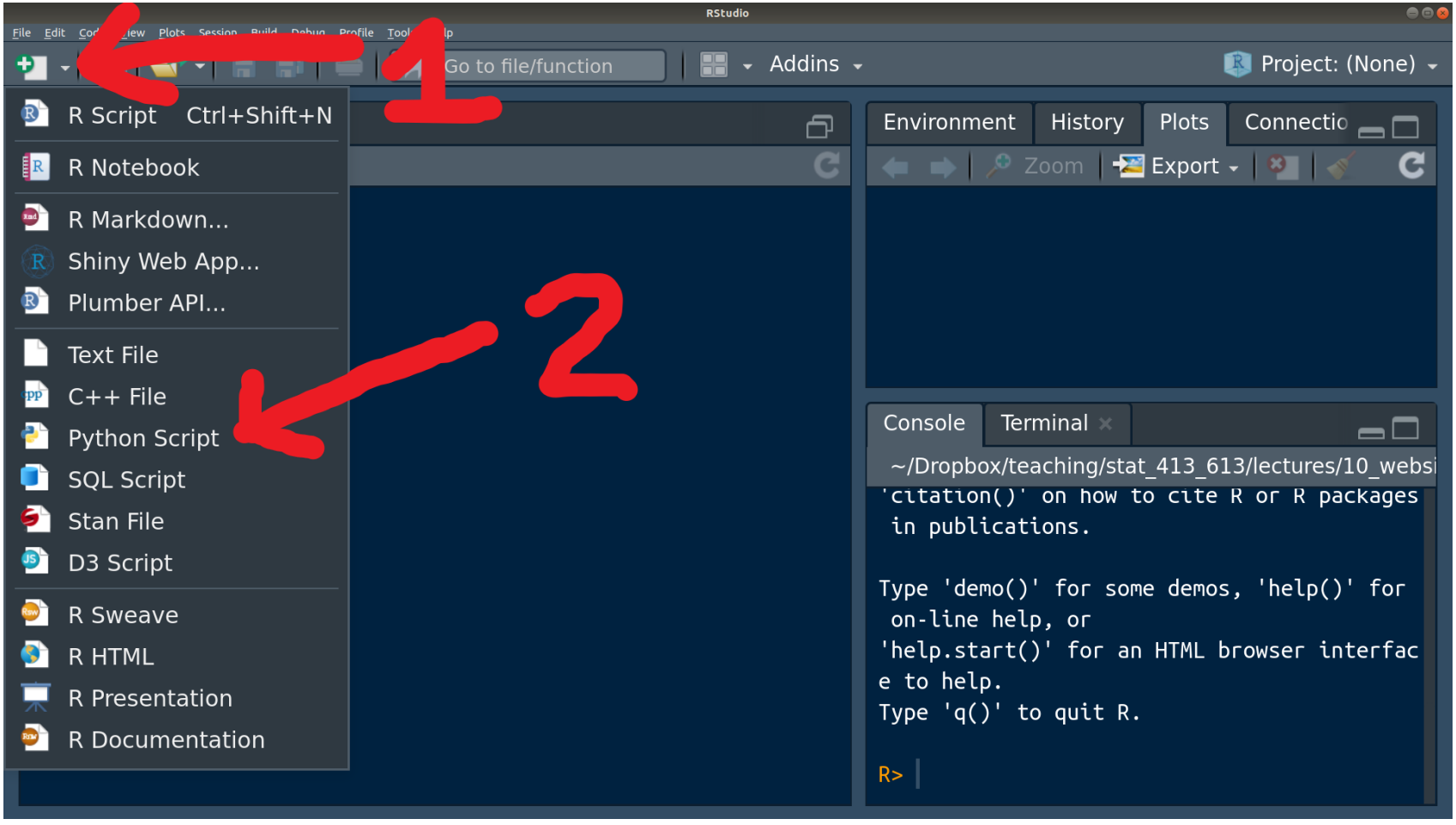
```
  py$x
```

```
[1]  8  9 11  3
```

- Sometimes it's buggy, but you can usually begin a Python REPL by also hitting Control/Command + Enter inside the Python chunk:



- Python scripts (where there is only Python code and no plain text) end in ".py". You can create a Python script in R Studio:



- Hitting Control/Command + Enter inside a Python script will also start a Python REPL.

Exercise: Create Fibonacci numbers

FIBONACHI

Python

```
num = 15
p = []
a,b = 0, 1
while b < num:
    a, b = b, a+b
    p.append(a)
print("The fibonacci numbers are:", p)
```

```
The fibonacci numbers are: [1, 1, 2, 3, 5, 8, 13]
```

```
num = 6
p = []
a,b = 0, 1
while b < num:
    a, b = b, a+b
    p.append(a)
    print("The growth of series in each step", p)
```

```
The growth of series in each step [1]
The growth of series in each step [1, 1]
The growth of series in each step [1, 1, 2]
The growth of series in each step [1, 1, 2, 3]
The growth of series in each step [1, 1, 2, 3, 5]
```