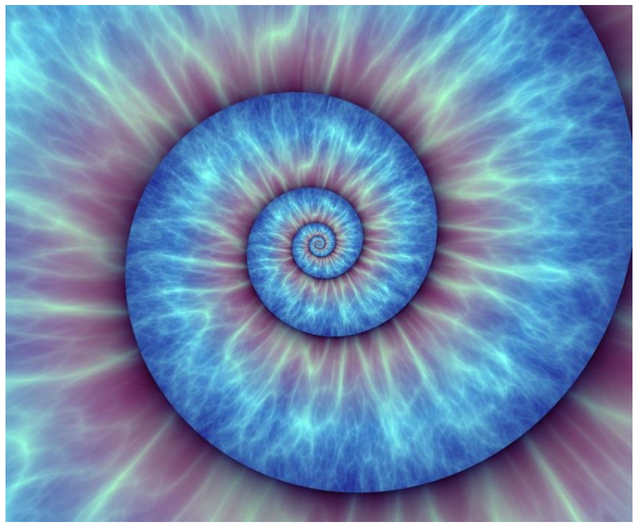
Fibonacci Sequences for Fun & Profit

Brian Spiering

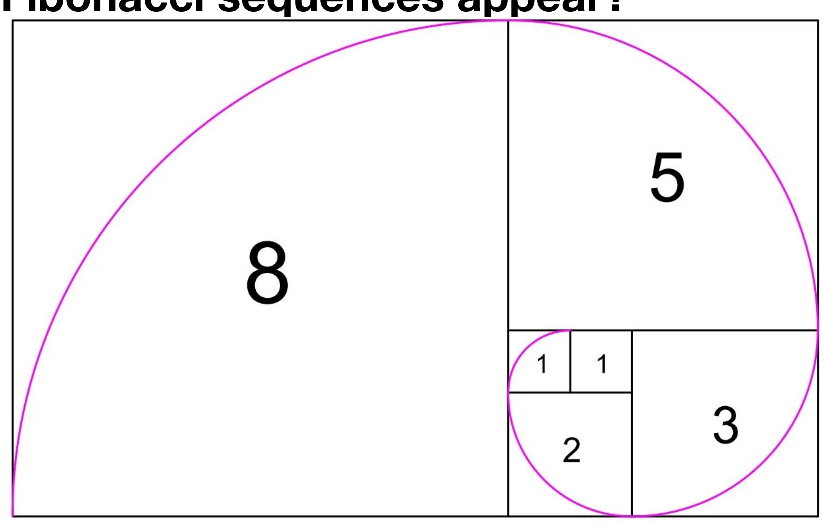


What is a Fibonacci sequence?

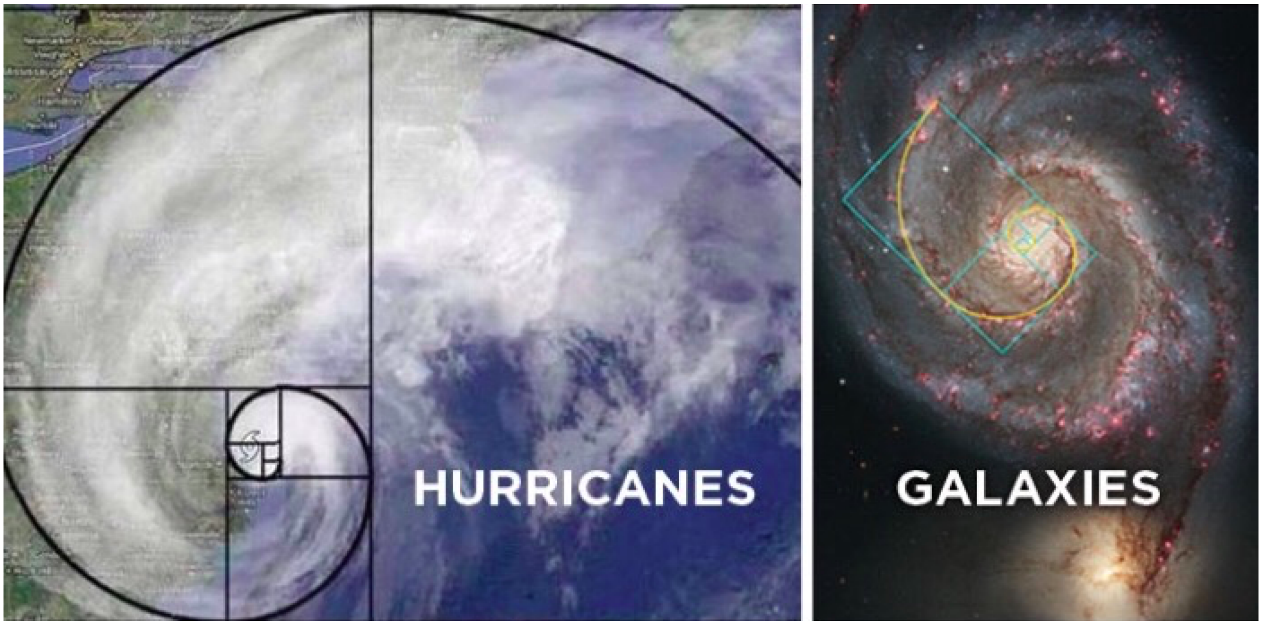
0, 1, 1, 2, 3, 5, 8, 13

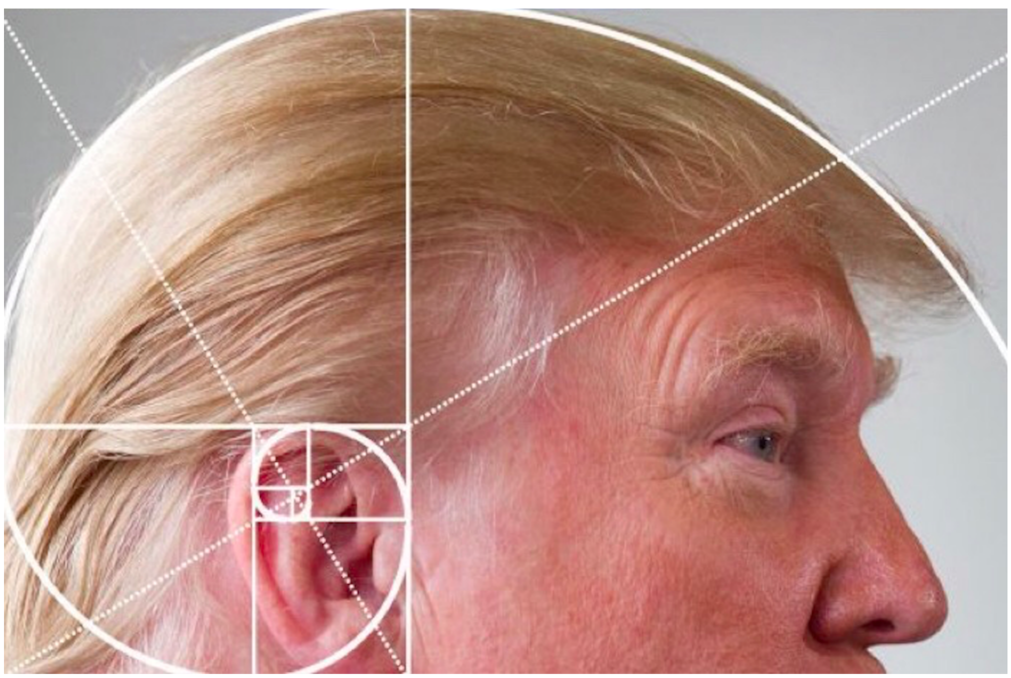
```
0 + 1 = 1
    1 + 1 = 2
        1 + 2 = 3
            2 + 3 = 5
                3 + 5 = 13
```

Where do Fibonacci sequences appear?



Golden Spiral 🐚





What are the ways to calculate a Fib sequence in Python 🐍?

- Recursion
- Dynamic Programming
- Imperative
- Closed form
- Functional

In [46]:
```python
def test_fib(f, test_large=False):
    "Test that a Fibonacci function returns the correct value"
    fib_sequence = {0: 0, # Index -> Fib value
                    1: 1,
                    2: 1,
                    3: 2,
                    4: 3,
                    9: 34,
                    25: 75_025
                   }

    for key, fib_value in fib_sequence.items():
        assert f(key) == fib_value

    if test_large:
        assert f(32) == 2_178_309

    print('tests pass 😁')
```

In [35]:
```python
def fib_recursive(n_th):
    "Calculate nth Fibonacci number using recursion"
    if n_th == 0: return 0
    if n_th == 1: return 1
    return fib_recursive(n_th-1) + fib_recursive(n_th-2)
```

In [36]:
```python
# Take a peek to be sure it is correct 👀
[fib_recursive(n_th) for n_th in range(10)]
```

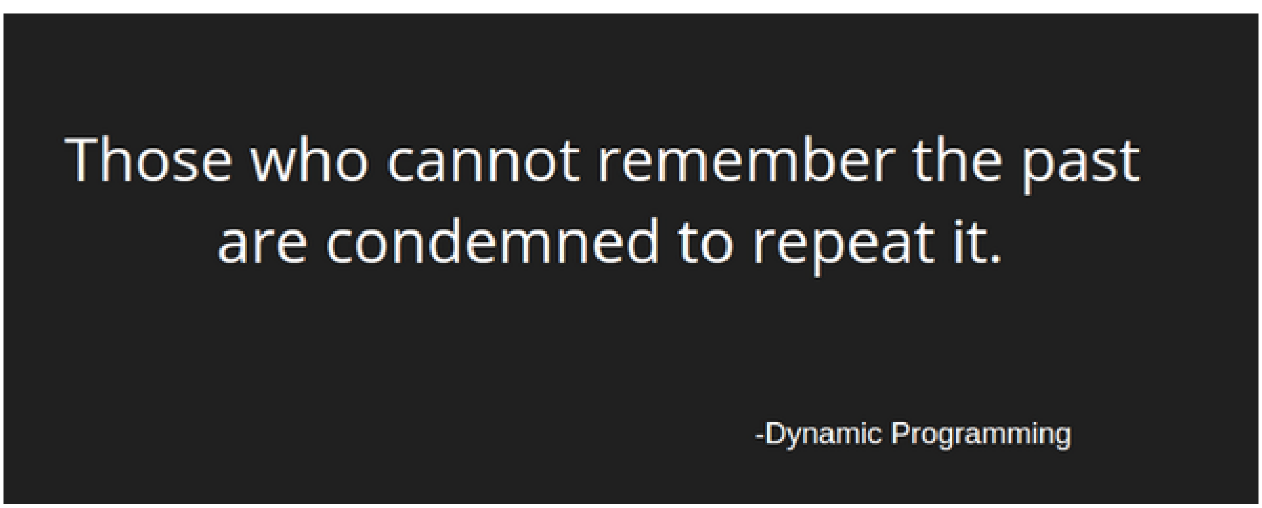
Out[36]:
```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

In [37]:
```python
# Let's test it
test_fib(fib_recursive)
```

```
tests pass 😁
```

In [38]:
```python
# Recursion does not scale in Python ⏳
test_fib(fib_recursive, test_large=True)
```

```
tests pass 😁
```



In [39]:
```
from functools import lru_cache as memo

@memo(maxsize=32)
def fib_cache(n_th):
    "Calculate nth Fibonacci number using dynamic programming"
    if n_th == 0: return 0
    if n_th == 1: return 1
    return fib_recursive(n_th-1) + fib_recursive(n_th-2)

test_fib(fib_cache)
```

tests pass 😁

In [40]:
```
test_fib(fib_cache, test_large=True)
```

tests pass 😁

In [41]:
```
def fib_subproblems(n_th):
    "Calculate nth Fibonacci number by storing the previous values"
    fib_seq = [0, 1]
    for i in range(n_th):
        fib_seq.append(fib_seq[-1]+fib_seq[-2])
    return fib_seq[-2]

test_fib(fib_subproblems, test_large=True)
```

tests pass 😁

Big O Analysis of Imperative Fib Function

Time complexity is O(n)

Space complexity is O(n) 👈 😱

In [28]:
```
def fib_swap(n_th):
    "Calculate nth Fibonacci number only keeping the needed values"
    a, b = 0, 1
    for _ in range(n_th):
        a, b = b, a+b
    return a

test_fib(fib_swap, test_large=True)
```

tests pass 😁

What is the best Big 0 for time complexity?

O(1) - Constant time

Constant time for mathematical operations are the closed-form solutions

Binet's Formula

$$F_n = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n$$

MathOps FTW🤘!

In [29]:
```
from math import sqrt

def fib_binet(n_th):
    "Calculate nth Fibonacci number using Binet's formula"
    first_term  = (1/sqrt(5))*((1+sqrt(5))/2)**n_th
    second_term = (1/sqrt(5))*((1-sqrt(5))/2)**n_th
    return round(first_term - second_term)

test_fib(fib_binet, test_large=True)
```

```
tests pass 😁
```

Fibonacci the Functional Way 🧙

In [2]:
```
def fib_gen():
    "A Fibonacci sequence generator"
    a, b = 0, 1
    while True:
        a, b = b, a+b
        yield a # Replace return statement
```

In [5]:
```python
from itertools import islice

n_th = 1_000  # 1_000, 10_000, 100_000
big_fib = next(islice(fib_gen(), n_th-1, n_th))
print(f"{big_fib:,}")
```

```
2,597,406,934,722,172,416,615,503,402,127,591,541,488,048,538,651,769,6
58,472,477,070,395,253,454,351,127,368,626,555,677,283,671,674,475,463,
758,722,307,443,211,163,839,947,387,509,103,096,569,738,218,830,449,30
5,228,763,853,133,492,135,302,679,278,956,701,051,276,578,271,635,608,0
73,050,532,200,243,233,114,383,986,516,137,827,238,124,777,453,778,337,
299,916,214,634,050,054,669,860,390,862,750,996,639,366,409,211,890,12
5,271,960,172,105,060,300,350,586,894,028,558,103,675,117,658,251,368,3
77,438,684,936,413,457,338,834,365,158,775,425,371,912,410,500,332,195,
991,330,062,204,363,035,213,756,525,421,823,998,690,848,556,374,080,17
9,251,761,629,391,754,963,458,558,616,300,762,819,916,081,109,836,526,3
52,995,440,694,284,206,571,046,044,903,805,647,136,346,033,000,520,852,
277,707,554,446,794,723,709,030,979,019,014,860,432,846,819,857,961,01
5,951,001,850,608,264,919,234,587,313,399,150,133,919,932,363,102,301,8
64,172,536,477,136,266,475,080,133,982,431,231,703,431,452,964,181,790,
051,187,957,316,766,834,979,901,682,011,849,907,756,686,456,845,066,28
7,392,485,603,914,047,605,199,550,066,288,826,345,877,189,410,680,370,0
91,879,365,001,733,011,710,028,310,473,947,456,256,091,444,932,821,374,
855,573,864,080,579,813,028,266,640,270,354,294,412,104,919,995,803,13
1,876,805,899,186,513,425,175,959,911,520,563,155,337,703,996,941,035,5
```

Takeaways 📝

- There are often many ways to compute values (with tradeoffs)
- A little bit of math helps
- Idiomatic Python can be memory efficient and fast (enough)

The End 🛑

[github.com/brianspiering/fibonacci_sequences (https://github.com/brianspiering/fibonacci_sequences)](https://github.com/brianspiering/fibonacci_sequences)

Benchmarking

In [13]:
```python
n = 30
```

In [14]:
```python
# How fast is the recursive implementation?
%timeit -n5 fib_recursive(n)
```

```
471 ms ± 23.9 ms per loop (mean ± std. dev. of 7 runs, 5 loops each)
```

In [15]:

```
# How fast is the dynamic programming implementation?
%timeit -n5 fib_cache(n)
```

```
449 ns ± 140 ns per loop (mean ± std. dev. of 7 runs, 5 loops each)
```

In [16]:

```
# How fast is the swap implementation?
%timeit -n5 fib_swap(n)
```

```
4.66 µs ± 232 ns per loop (mean ± std. dev. of 7 runs, 5 loops each)
```

Let's benchmark for big ns!

In [17]:

```
n = 100_000
```

In [18]:

```
# How fast is the swap implementation?
%timeit -n5 fib_swap(n)
```

```
161 ms ± 3.28 ms per loop (mean ± std. dev. of 7 runs, 5 loops each)
```

In [19]:
```
# How fast is the closed form implementation?
%timeit -n5 fib_binet(n)
```

```
---------------------------------------------------------------------------
--
OverflowError                             Traceback (most recent call las
t)
<ipython-input-19-1ffb82ce27c6> in <module>()
      1 # How fast is the closed form implementation?
----> 2 get_ipython().run_line_magic('timeit', '-n5 fib_binet(n)')

~/miniconda3/envs/ml/lib/python3.7/site-packages/IPython/core/interactive
shell.py in run_line_magic(self, magic_name, line, _stack_depth)
   2129                 kwargs['local_ns'] = sys._getframe(stack_depth).f
_locals
   2130             with self.builtin_trap:
-> 2131                 result = fn(*args,**kwargs)
   2132             return result
   2133 

<decorator-gen-61> in timeit(self, line, cell, local_ns)

~/miniconda3/envs/ml/lib/python3.7/site-packages/IPython/core/magic.py in
 <lambda>(f, *a, **k)
    185     # but it's overkill for just that one bit of state.
    186     def magic_deco(arg):
--> 187         call = lambda f, *a, **k: f(*a, **k)
    188 
    189         if callable(arg):

~/miniconda3/envs/ml/lib/python3.7/site-packages/IPython/core/magics/exec
ution.py in timeit(self, line, cell, local_ns)
   1099                     break
   1100 
-> 1101         all_runs = timer.repeat(repeat, number)
   1102         best = min(all_runs) / number
   1103         worst = max(all_runs) / number

~/miniconda3/envs/ml/lib/python3.7/timeit.py in repeat(self, repeat, numb
er)
    202         r = []
    203         for i in range(repeat):
--> 204             t = self.timeit(number)
    205             r.append(t)
    206         return r

~/miniconda3/envs/ml/lib/python3.7/site-packages/IPython/core/magics/exec
ution.py in timeit(self, number)
    157         gc.disable()
    158         try:
--> 159             timing = self.inner(it, self.timer)
    160         finally:
    161             if gcold:

<magic-timeit> in inner(_it, _timer)

<ipython-input-10-0b4fd556e558> in fib_binet(n)
```

```
      3 def fib_binet(n):
      4     "Calculate nth Fibonacci number using Binet's formula"
----> 5     first_term  = (1/sqrt(5))*((1+sqrt(5))/2)**n
      6     second_term = (1/sqrt(5))*((1-sqrt(5))/2)**n
      7     return round(first_term - second_term)

OverflowError: (34, 'Result too large')
```

In [ ]:

```
# How fast is the functional implementation?
%timeit -n5 next(islice(fib_gen(), n-1, n))
```