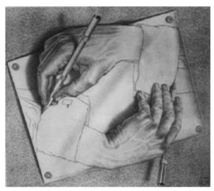
توابع بازگشتی

- تابع بازگشتی. تابعی که خودش را به صورت مستقیم یا غیرمستقیم فراخوانی می‌کند.

- مزایای یادگیری توابع بازگشتی.
  - آشنایی با یک سبک جدید تفکر (تفکر بازگشتی)
  - آشنایی با یک الگوی قدرتمند برنامه‌نویسی



- رابطه نزدیک با استقرای ریاضی.

فاکتوریل

- محاسبه فاکتوریل **n**.

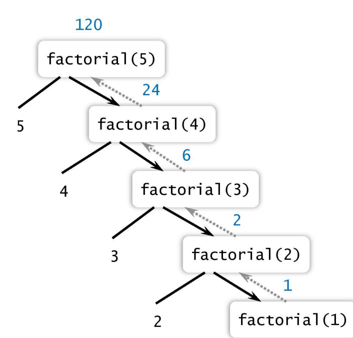
```
n! = 1 * 2 * ... * (n - 1) * n
```

$$n! = \begin{cases} 1 & n = 1 \\ n \times (n-1)! & n > 1 \end{cases}$$

```
def factorial(n):
    if n == 1: return 1
    return n * factorial(n - 1)
```

ختم بازگشت

فراخوانی بازگشتی



فاکتوریل

□ محاسبه فاکتوریل n.

```
n! = 1 * 2 * ... * (n - 1) * n
```

$$n! = \begin{cases} 1 & n = 1 \\ n \times (n-1)! & n > 1 \end{cases}$$

```
def factorial(n):
    if n == 1: return 1
    return n * factorial(n - 1)
```

ختم بازگشت: `if n == 1: return 1`

فراخوانی بازگشتی: `factorial(n - 1)`

```
factorial(5)
    factorial(4)
        factorial(3)
            factorial(2)
                factorial(1)
                    return 1
                return 2*1 = 2
            return 3*2 = 6
        return 4*6 = 24
    return 5*24 = 120
```

In [ ]:

```
'''
iterative:
            n! = 1*2*3*...*n

recursive:
            n! = n * (n-1)!
            1! = 1


3! = 3 * 2! = 3 * 2 = 6
2! = 2 * 1! = 2 * 1 = 2
1! = 1

'''
```

In [ ]:

```
def fact(n):
    f = 1
    if n == 0 :
        print('1')
    else:
        for i in range(1, n+1):
             f *= i
        print(f)

fact(60)   # 6
```

In [ ]:
```
def fact_rec(n):
    if n == 1:
        return 1
    else:
        return n * fact_rec(n-1)

print(fact_rec(60))     # 6
```

In [ ]:
```
'''
2*3 = 2 + (2*2) = 2 + 4 = 6
2*2 = 2 + (2*1) = 2 + 2 = 4
2*1 = 2
'''
def mul(x,y):
    if y == 0:
        return 0
    elif y == 1:
        return x
    else:
        return x + mul(x,y-1)

print(mul(2,3))    #6
```

In [ ]:
```
'''
2 ** 3 =  2 * (2**2) = 2 * 4 = 8
2 ** 2 =  2 * (2**1) = 2 * 2 = 4
2 ** 1 =  2
'''
def pow_rec(x,y):
    if y == 0:
        return 1
    elif x == 0:
        return 0
    elif y == 1:
        return x
    else:
        return x * pow_rec(x,y-1)

print(pow_rec(3,2))  # 9
```

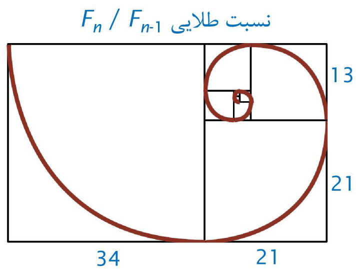
اعداد فیبوناچی

لئوناردو فیبوناچی
(۱۱۷۰ – ۱۲۵۰)

- دنباله فیبوناچی. فرض کنید به ازای $n > 1$ تعریف کنیم $F_n = F_{n-1} + F_{n-2}$ و $F_0 = 0$ و $F_1 = 1$.

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	...
F_n	0	1	1	2	3	5	8	13	21	34	55	89	144	233	...

- این دنباله پدیده‌های بسیاری را مدل‌سازی می‌کند و به طور گسترده‌ای در هنر و معماری یافت می‌شود.
  - مدلی برای نرخ تکثیر خرگوش‌ها
  - پوسته ملوانک (ناتیلوس)
  - مونالیزا



- دانستنی‌ها.
  - نسبت F_n به F_{n-1} به مقدار ϕ میل می‌کند.
  - مقدار F_n نزدیک‌ترین عدد صحیح به $\phi^n/\sqrt{5}$ است.

In [ ]:

```python
# fibonacci (10) : 0, 1, 1, 2, 3, 5, 8

def fibo(n):
    r = []
    a = 0
    b = 1
    while a < n:
        r.append(a)
        a, b = b, a+b
    return r

print(fibo(10))  # [0, 1, 1, 2, 3, 5, 8]
```

محاسبه اعداد فیبوناچی

- پرسش. [یک فرد کنجکاو] مقدار دقیق F_{60} چقدر است؟
- پاسخ. [یک برنامه‌نویس تازه‌کار] چند لحظه به من فرصت بده تا یک برنامه بازگشتی بنویسم.

```python
import sys

def fib(n):
    if n == 0: return 0
    if n == 1: return 1
    return fib(n-1) + fib(n-2)

n = int(sys.argv[1])
print(fib(n))
```

```
% python fibonacciR.py 5
5
% python fibonacciR.py 10
55
% python fibonacciR.py 12
144
% python fibonacciR.py 50
12586269025
% python fibonacciR.py 60
```

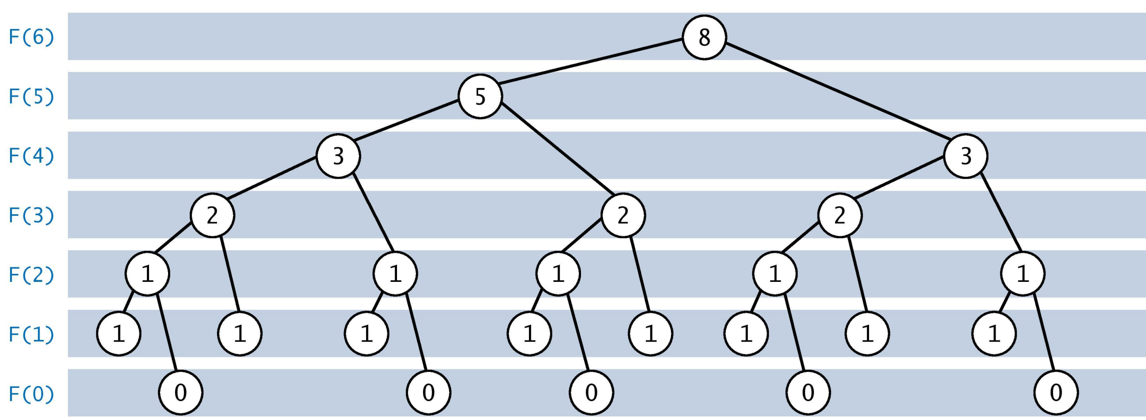
چند دقیقه طول می‌کشد. چرا؟

آیا رایانه من ایرادی پیدا کرده است؟

In [ ]:
```python
def fibo(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibo(n-1) + fibo(n-2)

print(fibo(4))   # 3
```

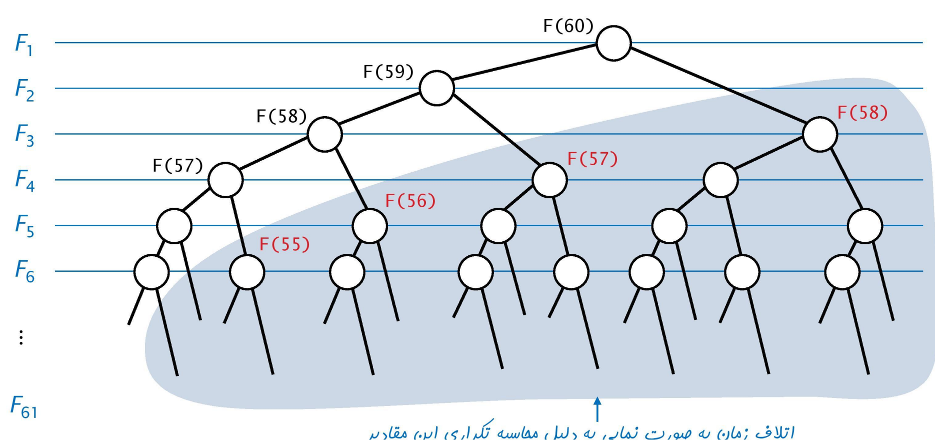
درخت فراخوانی بازگشتی برای اعداد فیبوناچی



درخت فراخوانی بازگشتی برای اعداد فیبوناچی

□ فرض کنید C_n بیانگر تعداد دفعات فراخوانی F(n) در هنگام محاسبه F(60) باشد.

n	C_n
60	1
59	1
58	2
57	3
56	5
55	8
⋮	⋮
0	$>2.5\times10^{12}$



اجتناب از اتلاف نمایی زمان

```
import sys

memo = [0] * 200

def fib(n):
    if n == 0: return 0
    if n == 1: return 1
    if memo[n] == 0:
        memo[n] = fib(n-1) + fib(n-2)
    return memo[n]

n = int(sys.argv[1])
print(fib(n))
```

- یادداشت‌برداری.
  - استفاده از یک آرایه به منظور به خاطر سپاری تمامی مقادیر محاسبه شده
  - اگر مقداری قبلا محاسبه شده، آن را برگردان.
  - در غیر این صورت، آن مقدار را محاسبه کن، در آرایه ذخیره کن و سپس آن را برگردان.

```
% python fibonacciM.py 60
1548008755920
% python fibonacciM.py 80
23416728348467685
% python fibonacciM.py 100
354224848179261915075
```

In [ ]:

```python
memo = [0] * 200

def fib(n):
    if n == 0:
        return 0
    if n == 1:
        return 1
    if memo [n] == 0:
        memo[n] = fib(n-1) + fib(n-2)
    return memo[n]

print(fib(60))
```

In [ ]:

```python
def f(lst):
    if len(lst) == 1:
        return lst[0]
    else:
        return lst[0] + f(lst[1:])

a = [2, 4, 5, 6, 7]
print(f(a))   # 24

'''
f([2, 4, 5, 6, 7]) = 2 + f([4, 5, 6, 7]) =2 + 22 = 24
f([4, 5, 6, 7])    = 4 + f([5, 6, 7]) = 4 + 18 = 22
f([5, 6, 7])       = 5 + f([6, 7]) = 5 + 13 = 18
f([6, 7])          = 6 + f([7])  = 6 + 7 = 13
f([7])             = 7
'''
```

In [ ]:
```
def sum_digits(n):
    if n == 0:
        return 0
    else:
        return n % 10 + sum_digits(int(n/10))

print(sum_digits(345))   # 345 = 3+4+5=12

'''
sum_digits(345) = 5 + sum_digits(34)= 5 + 7 = 12
sum_digits(34)  = 4 + sum_digits(3) = 4 + 3 = 7
sum_digits(3)   = 3 + sum_digits(0) = 3 + 0 = 3
'''
```

In [ ]:
```
# n + (n-2) +(n-4) +...

def sum_series(n):
    if n < 1 :
        return 0
    else:
        return n + sum_series(n-2)

print(sum_series(10))     # 10 + 8 + 6 + 4 + 2 = 30
```

In [ ]:
```
def f(n,base):
     s = '0123456789ABCDEF'
     if n < base:
         return s[n]
     else:
         return f(n//base , base) + s[ n % base]

print(f(10,16))       # A
print(f(25,16))       # 19

'''
f(25,16) = f(1,16) +s[9] = 1+9 = 19
f(1,16)  = s[1]  = 1

'''


print(f(8,2))         # 1000
'''
f(8,2) = f(4,2) + s[0] = 100 + 0 = 1000
f(4,2) = f(2,2) + s[0] = 10  + 0 = 100
f(2,2) = f(1,2) + s[0] = 1   + 0 = 10
f(1,2) = s[1] =1
'''


print(f(16,16))         # 10

print(f(129,2))         # 10000001
```

In [ ]:
```python
def binary_search(lst, x, start=0, end=None):
    if end is None:
        end = len(lst) - 1
    if start > end:
        return False
    mid = (start + end) // 2
    if x == lst[mid]:
        return mid
    if x < lst[mid]:
        return binary_search(lst, x, start, mid - 1)
    return binary_search(lst, x, mid + 1, end)

a = [2, 4, 7, 12, 19, 25, 38]
print(binary_search(a, 19 ))    # 4
print(binary_search(a, 4 ))     # 1
print(binary_search(a, 20))     # False
```

توابع بازگشتی: خطاهای متداول (۱)

- فراموش کردن حالت پایه (ختم بازگشت).
  - فراموش کردن حالت پایه، منجر به گیر افتادن در یک حلقه بی‌نهایت می‌شود.

```
def H(n):
    return H(n-1) + 1.0/n
```

- عدم تضمین همگرایی.
  - استفاده از فراخوانی‌های بازگشتی برای حل زیرمسائلی که کوچک‌تر نیستند.

```
def H(n):
    if n == 1: return 1.0
    return H(n) + 1.0/n
```

توابع بازگشتی: خطاهای متداول (۲)

- محاسبات تکراری.
  - یک تابع بازگشتی ساده نیز ممکن است به دلیل محاسبات تکراری، دارای پیچیدگی زمانی نمایی باشد!

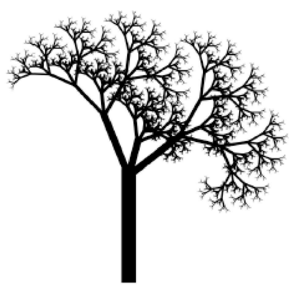
```
def fib(n):
    if n == 0: return 0
    if n == 1: return 1
    return fib(n-1) + fib(n-2)
```

$F_n = F_{n\text{-}1} + F_{n\text{-}2}$

```
fib(5)
    fib(4)
        fib(3)
            fib(2)
                fib(1)
                    return 1
                fib(0)
                    return 0
                return 1+0 = 1
            fib(1)
                return 1
            return 1+1 = 2
        fib(2)
            fib(1)
                return 1
            fib(0)
                return 0
            return 1+0 = 1
```

محاسبات تکراری

خلاصه



- چگونه می‌توان یک تابع بازگشتی نوشت؟
  - حالت پایه، فراخوانی بازگشتی.
  - ردیابی اجرای یک تابع بازگشتی.
  - استفاده از شکل.
- مزایای یادگیری توابع بازگشتی.
  - آشنایی با یک سبک جدید تفکر (تفکر بازگشتی)
  - آشنایی با یک الگوی قدرتمند برنامه نویسی
- تقسیم و حل. یک راه‌حل ظریف و زیبا برای بسیاری از مسائل مهم.
- برنامه‌ریزی پویا. اجتناب از محاسبات تکراری با حل زیرمسائل از پایین به بالا و ذخیره راه‌حل‌ها.

دانشگاه شهید مدنی آذربایجان
برنامه نویسی مقدماتی با پایتون
امین گلزاری اسکوئی
۱۴۰۰-۱۴۰۱

[Codes and Projects (click here) (https://github.com/Amin-Golzari-Oskouei/Python-Programming-Course-Basic-2021)](https://github.com/Amin-Golzari-Oskouei/Python-Programming-Course-Basic-2021) [slides and videos (click here) (https://drive.google.com/drive/folders/1ZsQjBJJ4UAAp9zrGxm3c4qrhnvGBUYHw)](https://drive.google.com/drive/folders/1ZsQjBJJ4UAAp9zrGxm3c4qrhnvGBUYHw)