
RECURSION

“THE PRACTICE OF COMPUTING USING PYTHON 3RD EDITION” BY WILLIAM PUNCH AND RICHARD ENBODY

OUTLINE

Apa itu Rekursi

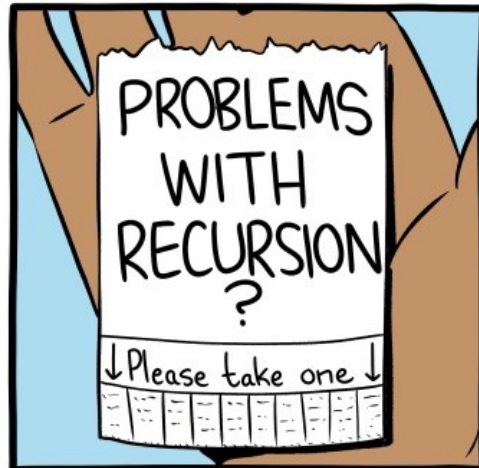
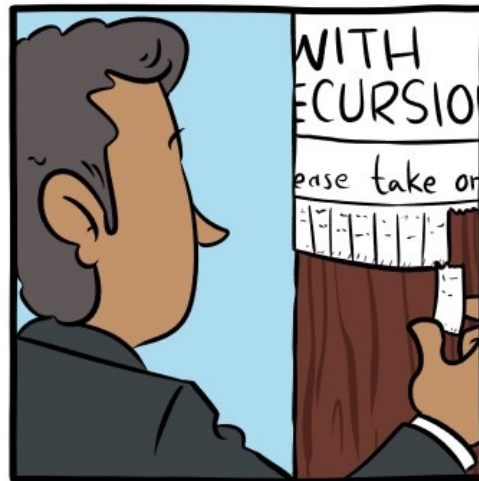
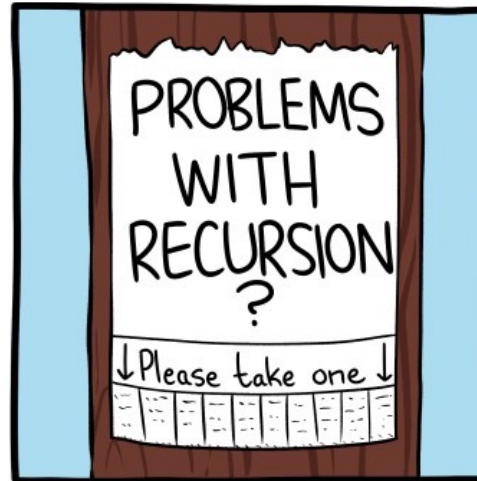
Contoh Rekursi

Bagaimana Cara Kerja Rekursi

Latihan Rekursi



to understand recursion you must first understand recursion





Patrick Infinite Recursive 😊

Apa sih rekursif itu?

Rekursif (*Recursion*) adalah fungsi yang memanggil dirinya sendiri

```
def f (n) :  
    ...  
    f (n-1)
```

Bagaimana kita bisa memahami rekursif?

2 hal yang perlu kita pahami ketika kita ingin menggunakan rekursif dalam memecahkan permasalahan yang ada:

- Bagaimana kita bisa memecah permasalahan besar dengan cara memecahnya menjadi permasalahan yang lebih kecil. Nantinya kita cukup memecahkan permasalahan kecil ini saja.
- Mengetahui kapan kita bisa menyatakan bahwa permasalahan yang kita hadapi sudah yang paling terkecil

Studi Kasus: Mencetak Langkah

Andai kita ingin membuat sebuah program yang menceritakan *quote* berikut

“a journey of a thousand steps begins with a single step. ”

Notes: Jika kita sudah mencapai Langkah ke-999, maka kita bisa dengan mudah mencapai Langkah ke-1000. Hal ini juga berlaku ketika kita mencapai Langkah ke-999, maka kita bisa dengan mudah mencapai Langkah ke-998. Proses ini terus berlangsung secara rekursif hingga nantinya kita mencapai Langkah pertama dan kita mengetahui apa yang akan kita lakukan pada setiap langkahnya.

Code Listing 15.1

```
def take_step(n):  
    if n == 1: # base case  
        return "Easy"  
    else:  
        this_step = "step(" + str(n) + ")"  
        previous_steps = take_step(n-1) # recursive call  
        return this_step + " + " + previous_steps
```

In [1]: takeStep(4)

Out [1]: 'step(4) + step(3) + step(2) + Easy'

Faktorial

Dalam matematika, sebuah fungsi faktorial dapat didefinisikan sebagai berikut:

$$n! = n \times (n - 1) \times (n - 2) \times \dots \times 2 \times 1$$

Contoh:

$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$$

Faktorial

Jika kita menggunakan rekursif, maka fungsi faktorial dapat kita definisikan sebagai berikut: *

$$n! = n \times (n - 1)!$$

Contoh:

$$5! = 5 \times 4!$$

* catatan: $0! = 1$ and $1! = 1$

Yuk, kita implementasikan faktorial dalam Python

```
def factorial(num): #header of factorial function  
  
    # 0! Or 1! return 1  
    if (num == 0) or (num == 1):  
        return 1
```

Yuk, kita implementasikan faktorial dalam Python (Lanj.)

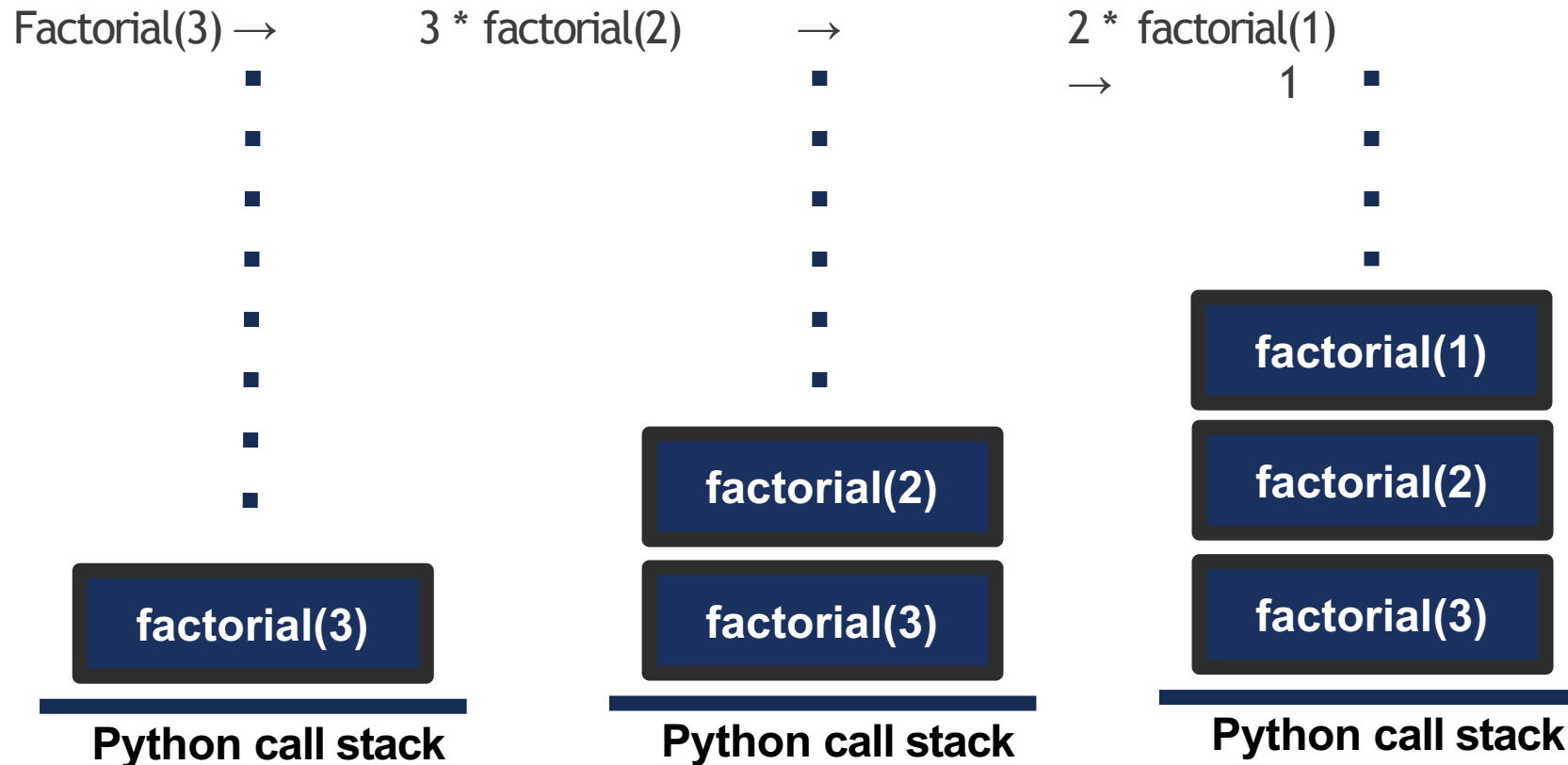
```
def factorial(num): #header of factorial function

    # BASE CASE: 0! Or 1! return 1
    if (num == 0) or (num == 1):
        return 1

    # RECURSIVE CASE: if num > 1, use recursion
    return num * factorial(num-1)
```

Simulasi 3! Pada Program Python kita!

□ Calling trace:



Simulasi 3! Pada Program Python kita!

Return trace:

1

->

2 * 1

->

3 * 2

■
■
■
■
■
■
■

1

factorial(2)

factorial(3)

Python call stack

■
■
■
■
■
■
■
■
■
■
■

2

factorial(3)

Python call stack

■
■
■
■
■
■
■
■
■
■
■
■

6

Python call stack

Komponen utama dalam rekursif

```
def factorial(num):  
    # BASE CASE  
    if (num == 0) or (num == 1):  
        return 1  
  
    # RECURSION CASE  
    return num * factorial(num-1)
```

- **Base case**
Kasus dimana rekursif akan berhenti
- **Recursion case**
Kasus dimana kita akan memecah permasalahan menjadi lebih kecil

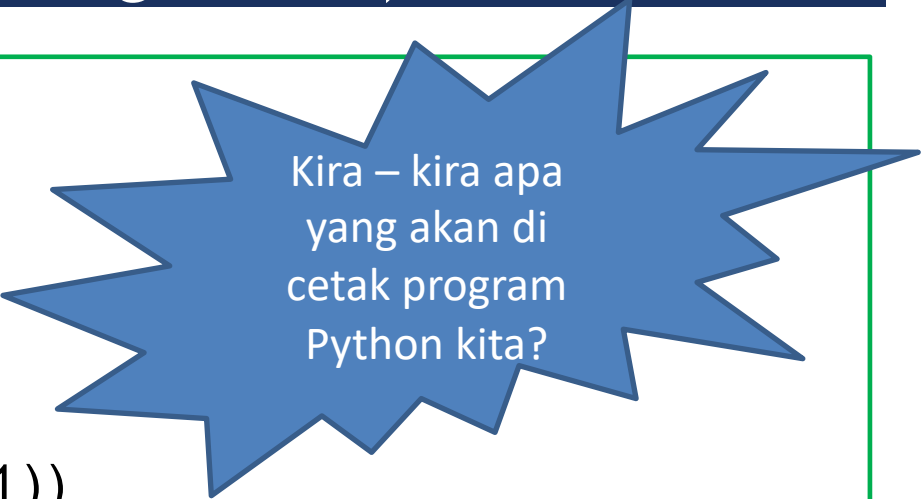
Implementasi faktorial dalam Python (*Debug Mode*)

```
def factorial(num): # header of factorial function

    print("Calculate the factorial of ", num)

    # BASE CASE: 0! or 1! return 1
    if (num == 0) or (num == 1):
        print("Factorial of {} is {}".format(num, 1))
        return 1

    # RECURSION CASE: for num > 1, use recursive
    factorial_val = num * factorial(num-1)
    print("Factorial of {} is {}".format(num, factorial_val))
    return factorial_val
```



Kira – kira apa
yang akan di
cetak program
Python kita?

Rekursif untuk implementasi Bilangan Fibonacci

```
def fibonacci(n):  
    """Recursive Fibonacci sequence ."""  
    if n == 0 or n == 1: # base cases  
        return 1  
    else:  
        return fibonacci(n-1) + fibonacci(n-2) # recursive case
```

Yuk Berlatih! Lengkapi Program dibawah ini...

```
# recursive function untuk membalikan string
# abcde -> edcba
# maman -> namam
# macan -> nacam

def reverser (a_str):
    # base case
    # recursive step
        # divide into parts
        # conquer/reassemble

the_str = input("Reverse what string:")
result = reverser(the_str)
print("The reverse of {} is {}".format(the_str,result))
```

REVERSING A STRING - Contoh Solusi

```
# Reverse a string using a recursive function .
def reverse (a_str):
    """Recursive function to reverse a string ."""
    print("Got as an argument:",a_str) # base case
    if len(a_str) == 1:
        print("Base Case!")
        return a_str # recursive step
    else:
        new_str = reverse(a_str[1:]) + a_str[0]
        print("Reassembling {} and {} into {}".format(a_str[1:],a_str[0], new_str))
        return new_str

the_str = input("What string: ")
print()
result_str = reverse(the_str)
print("The reverse of {} is {}".format(the_str,result_str))
```



Bagaimana Cara Kerja Rekursi?



Struktur Data Stack(1/3)



- Komputer sebenarnya menghafal urutan fungsi yang dipanggil dalam sebuah struktur data yang dinamakan *stack* (tumpukkan)
- Tentunya ketika kita ingin mengambil benda dalam tumpukkan, kita akan ambil dari yang paling atas terlebih dahulu. Dalam tumpukkan, benda paling atas tentunya adalah benda yang disimpan paling akhir. Disini dikenal istilah namanya LIFO(**LIFO : Last In, First Out**)
- Dalam sebuah struktur data *stack*, dikenal 3 operasi umum: **pop**, **push**, dan **top**.

Struktur Data Stack(1/3)

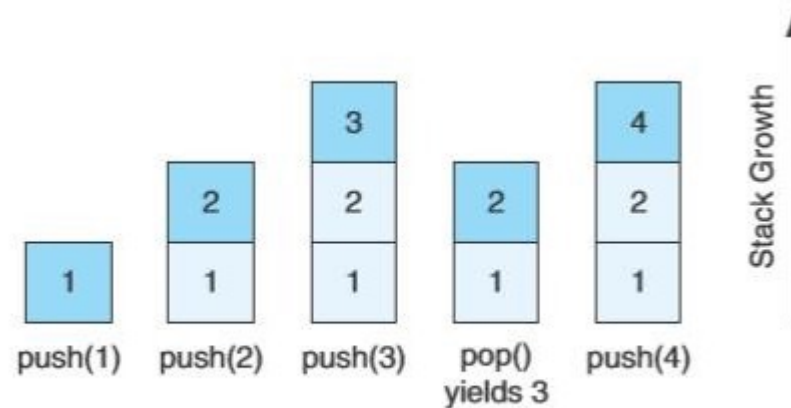


FIGURE 15.2 The operation of a stack data structure.

Stack Terminology	Python Terminology	Action
top()	List[-1]	Return the value of top of stack
push(x)	List.append(x)	Push x onto top of stack
y = pop()	y = List.pop()	Pop top off of stack and assign to y

TABLE 15.1 Stack terminology translated to Python.

Struktur Data Stack(1/3)

```
In [1]: stack_list = [1,2,3] # create a stack  
In [2]: x = stack_list.pop() # pop an item off the stack  
In [3]: x # the popped item was assigned to x  
Out [3]: 3
```

```
In [4]: stack_list # pop removed the item  
Out [4]: [1, 2]
```

```
In [5]: stack_list.append(7) # push 7 onto the stack (using append)  
In [6]: stack_list  
Out [6]: [1, 2, 7]
```

```
In [7]: stack_list[-1] # top()  
Out [7]: 7
```

```
In [8]: stack_list # top() doesn't change the stack  
Out [8]: [1, 2, 7]
```


Stacks dan kegunaannya dalam pemanggilan *Function*

Code Listing 15.7

```
def factorial(n):  
    """Recursive Factorial with print to show operation."""  
    indent = 4*(6-n)*" " # more indent on deeper recursion  
    print(indent + "Enter factorial n = ", n)  
    if n == 1: # base case  
        print(indent + "Base case.")  
        return 1  
    else: # recursive case  
        print(indent + "Before recursive call f(" + str(n-1) + ")")  
        # separate recursive call allows print after call  
        rest = factorial(n-1)  
        print(indent + "After recursive call f(" + str(n-1) + ") = ", rest)  
        return n * rest
```

Stacks dan kegunaannya dalam pemanggilan *Function* (Lanj.)

Code Listing 15.7

```
def factorial(n):  
    """Recursive Factorial with print to show operation."""  
    indent = 4*(6-n)*" " # more indent on deeper recursion  
    print(indent + "Enter factorial n = ", n)  
    if n == 1: # base case  
        print(indent + "Base case.")  
        return 1  
    else: # recursive case  
        print(indent + "Before recursive call f(" + str(n-1) + ")")  
        # separate recursive call allows print after call  
        rest = factorial(n-1)  
        print(indent + "After recursive call f(" + str(n-1) + ") = ", rest)  
        return n * rest
```

In [1]: factorial(4)

Out [1]: Enter factorial n = 4
Before recursive call f(3)
Enter factorial n = 3
Before recursive call f(2)
Enter factorial n = 2
Before recursive call f(1)
Enter factorial n = 1
Base case.
After recursive call f(1) = 1
After recursive call f(2) = 2
After recursive call f(3) = 6

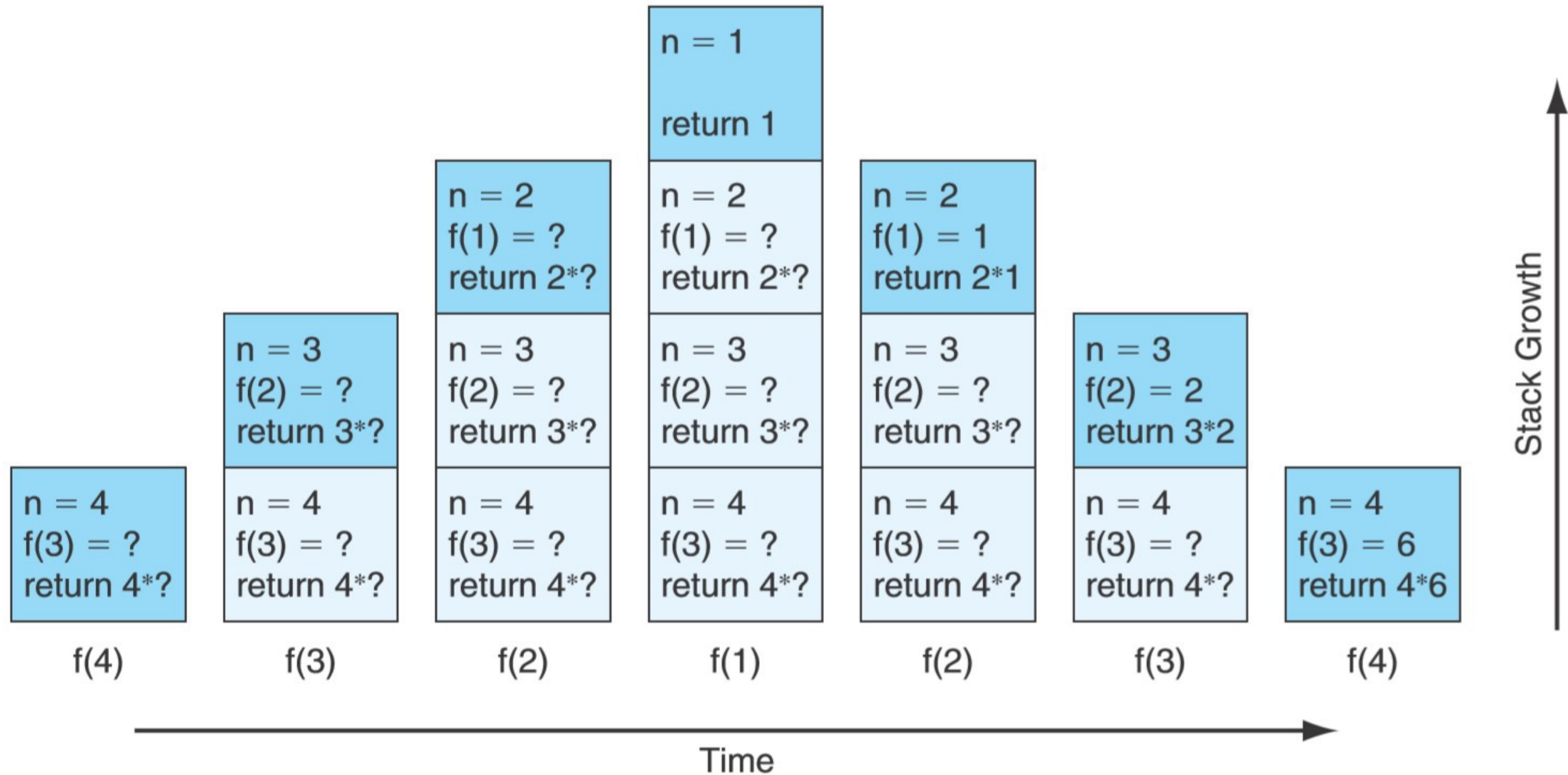


FIGURE 15.3 Call stack for `factorial(4)`. Note the question marks.

Implementasi Fungsi Fibonacci (II)

- Implementasi fungsi Fibonacci sebelumnya terlalu kompleks dan mahal dari segi komputasi.
- Gambar 15.4 menunjukkan bahwa nilai Fibonacci 3 dihitung 8 kali ketika menghitung nilai 8.
- Hal ini dapat kita perbaiki dengan mudah menggunakan **dictionary**. Andaikan kita mengingat setiap nilai yang dihitung dalam perhitungan Fibonacci. Pada implementasi baru ini, kita coba untuk selalu melihat **dictionary** terlebih dahulu sebelum melakukan perhitungan nilai Fibonacci. Hal ini, membantu kita untuk memastikan bahwa setiap nilai Fibonacci dihitung hanya 1 kali saja.

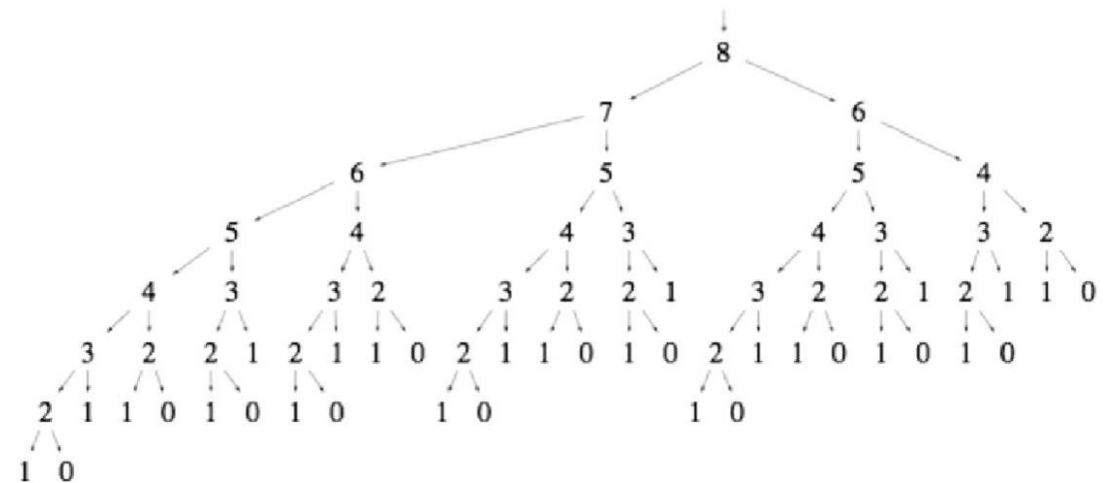


FIGURE 15.4 The simple Fibonacci code recalculates the same value.

Implementasi baru Fibonacci

Code Listing 15.8

```
def fibonacci(n):  
    """Recursive fibonacci that remembers previous values"""  
    if n not in fibo_dict:  
        # recursive case, store in the dict  
        fibo_dict[n] = fibonacci(n-1) + fibonacci(n-2)  
    return fibo_dict[n]  
  
# global fibonacci dictionary.  
fibo_dict = {}  
  
# enter the base cases  
fibo_dict[0] = 1  
fibo_dict[1] = 1  
  
fibo_val = input("Calculate what Fibonacci value:")  
print("Fibonnaci value of", fibo_val, "is",  
      fibonacci(int(fibo_val)))
```



EXERCISES



QUEST 1:

Apa yang dilakukan Fungsi berikut?(Asumsikan $NUM > 0$)

```
def mystery(num):  
    if num == 1:  
        return 1  
    return num + mystery(num-1)
```

QUEST 2:

Lengkapi fungsi perpangkatan berikut, dengan asumsi N tidak akan negatif

```
def power(num, n) :
```

```
.....
```


QUEST 2: Contoh Solusi

```
def power(num, n):  
    # base case: num powered by 0 = 1  
    if n == 0:  
        return 1  
  
    # recursion case  
    return num * power(num, n-1)
```

QUEST 3A: MAXIMUM_ITER(LST)

Jika kita asumsikan lst tidak kosong, carilah nilai paling tinggi dalam sebuah list yang di-*pass* dalam parameter lst*

```
def maximum_iter(lst):  
    ...
```

***) Buat dulu jawaban tanpa rekursif, baru dibuat versi rekursifnya**

QUEST 3A: MAXIMUM_ITER(LST)

Jika kita asumsikan lst tidak kosong, carilah nilai paling tinggi dalam sebuah list yang di-pass dalam parameter lst* (Versi Iterasi For)

```
def maximum_iter(lst):  
  
    max_temp = lst[0]  
  
    for i in range(1, len(lst)):  
        if lst[i] > max_temp:  
            max_temp = lst[i]  
  
    return max_temp
```

***) Buat dulu jawaban tanpa rekursif, baru dibuat versi rekursifnya**

QUEST 3B: MAXIMUM_REC(LST)

Jika kita asumsikan `lst` tidak kosong, carilah nilai paling tinggi dalam sebuah list yang di-*pass* dalam parameter `lst`*

```
def maximum_rec(lst):
```

```
    ...
```

QUEST 3B: MAXIMUM_REC(LST)

Jika kita asumsikan lst tidak kosong, carilah nilai paling tinggi dalam sebuah list yang di-pass dalam parameter lst* (Versi Rekursif)

```
def maximum_rec(lst):  
  
    # base case  
    if len(lst) == 1:  
        return lst[0]  
  
    # recursion case  
    max_rest = maksimum_rec(lst[1:])  
    if max_rest > lst[0]:  
        return max_rest  
    else:  
        return lst[0]
```

QUEST 3A&B: MAXIMUM_REC(LST)

Perbandingan kedua pendekatan

```
def maximum_iter(lst):  
  
    max_temp = lst[0]  
  
    for i in range(1, len(lst)):  
        if lst[i] > max_temp:  
            max_temp = lst[i]  
  
    return max_temp
```

**ITERATIVE
APPROACH**

```
def maximum_rec(lst):  
    # base case  
    if len(lst) == 1:  
        return lst[0]  
  
    # recursion case  
    max_rest = maximum_rec(lst[1:])  
    if max_rest > lst[0]:  
        return max_rest  
    else:  
        return lst[0]
```

**RECURSIVE
APPROACH**

QUEST 4: Buat sebuah fungsi untuk mencari apakah elemen ada didalam sebuah list menggunakan rekursif

□ Example:

```
is_in(5, [])  
>>> False  
is_in(5, [1,2,3,4,5])  
>>> True  
is_in(5, [1,2,3,4])  
>>> False  
is_in(5, [1,2,3,4,[5,6]])  
>>> True  
is_in(5, [1,2,[3,4,[5,6]]])  
>>> True  
is_in(5, [1,2,[3,4,[6]]])  
>>> False
```

QUEST 4: Buat sebuah fungsi untuk mencari apakah elemen ada didalam sebuah list menggunakan rekursif

```
def is_in(el, lst):  
  
    if len(lst) == 0:  
        return False  
  
    if len(lst) == 1:  
        if type(lst[0]) == list:  
            return is_in(el, lst[0])  
        else:  
            return el == lst[0]  
    else:  
        return is_in(el, lst[0:1]) or is_in(el, lst[1:])
```


QUEST 5: Buatlah sebuah fungsi untuk menghitung jumlah nilai dalam sebuah list (Diasumsikan list bukan sebuah nested list dan tidak kosong)

```
def total(lst):
```

```
    .....
```

QUEST 5: Buatlah sebuah fungsi untuk menghitung jumlah nilai dalam sebuah list (Diasumsikan list bukan sebuah nested list dan tidak kosong)

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

QUEST 6: Buatlah sebuah fungsi untuk memeriksa apakah sebuah string tergolong Palindrome dengan menggunakan rekursif

```
def palindrome(a_str):
```

```
    ...
```

QUEST 6: Buatlah sebuah fungsi untuk memeriksa apakah sebuah string tergolong Palindrome dengan menggunakan rekursif

```
def palindrome(a_str):  
    if len(a_str)==0:  
        return True  
    elif len(a_str)==1:  
        return True  
    else:  
        if a_str[0]==a_str[-1]:  
            return palindrome(a_str[1:-1])  
        else:  
            return False
```

QUEST 7: Apakah sebuah bilangan n adalah bulat ganjil atau genap?
(untuk setiap bilangan $n \geq 0$) Implementasi menggunakan rekursif

□ Any integer n is **even** if $n-1$ is **odd**

And

□ Any integer n is **odd** if $n-1$ is **even**

QUEST 7: Apakah sebuah bilangan n adalah bulat ganjil atau genap? (untuk setiap bilangan $n \geq 0$) Implementasi menggunakan rekursif

```
def is_even(n):
```

```
    .....
```

```
def is_odd(n):
```

```
    .....
```

QUEST 7: Apakah sebuah bilangan n adalah bulat ganjil atau genap? (untuk setiap bilangan $n \geq 0$) Implementasi menggunakan rekursif

```
def is_even(n):  
    if n == 0:  
        return True  
    else:  
        return is_odd(n-1)  
  
def is_odd(n):  
    if n == 0:  
        return False  
    else:  
        return is_even(n-1)
```

Apa yang kita bahas disini



- Rekursif adalah sebuah fungsi yang memanggil dirinya sendiri
recursive function calls itself
- Sebuah fungsi rekursif membutuhkan *base case* supaya *recursion case* tidak dijalankan selamanya ;)
- Setiap pemanggilan *recursion case*, memastikan bahwa permasalahan akan dipecah menjadi lebih kecil

CREDITS (IMAGES, ICONS, ETC)

- <https://prateekvjoshi.com/2013/10/05/understanding-recursion-part-i/>
- https://www.tutorialspoint.com/data_structures_algorithms/binary_search_tree.htm
- <https://www.petanikode.com/fungsi-rekursif/>

CREDITS (TEACHING KNOWLEDGE)

- [https://en.wikipedia.org/wiki/Recursion_\(computer_science\)](https://en.wikipedia.org/wiki/Recursion_(computer_science))
- <https://introcs.cs.princeton.edu/java/home/>
- All other sources that I probably forgot to mention