Ceng 111 – Fall 2020
Week 11

Recursion & Iteration

Credit: Some slides are from the “Invitation to Computer Science” book by G. M. Schneider, J. L. Gersting and some from the “Digital Design” book by M. M. Mano and M. D. Ciletti.

Today

- Midterm feedback
- Finalize recursion
- Iteration

Midterm feedback

- Interrupts, especially how interrupts are handled.
- The functionalities/roles of MBR, BIOS, ALU and CU.
- Recursion. In some recursion questions, you had difficulty tracing the code. But don't worry, we will spend more time and see more examples with recursion and hopefully, it will get better.
- Turing Machine.
- Questions related to basic data types in Python (switching to the long representation in v3, whether or not CPU decides to switch to integers for real numbers with zero fraction etc.).

Administrative Notes

- Live sessions schedule change
  - Tue 13:40 Session (a.k.a. common session)
  - Wed 10:40 Session
  - ~~Wed 15:40: Section 2~~
  - ~~Thu 15:40: Section 1~~
- Social session
- The labs
- Office hours: Tue 10:30
- Lab Exam 1: 23 Dec
- Final: 30 January 13:30

When to avoid recursion!

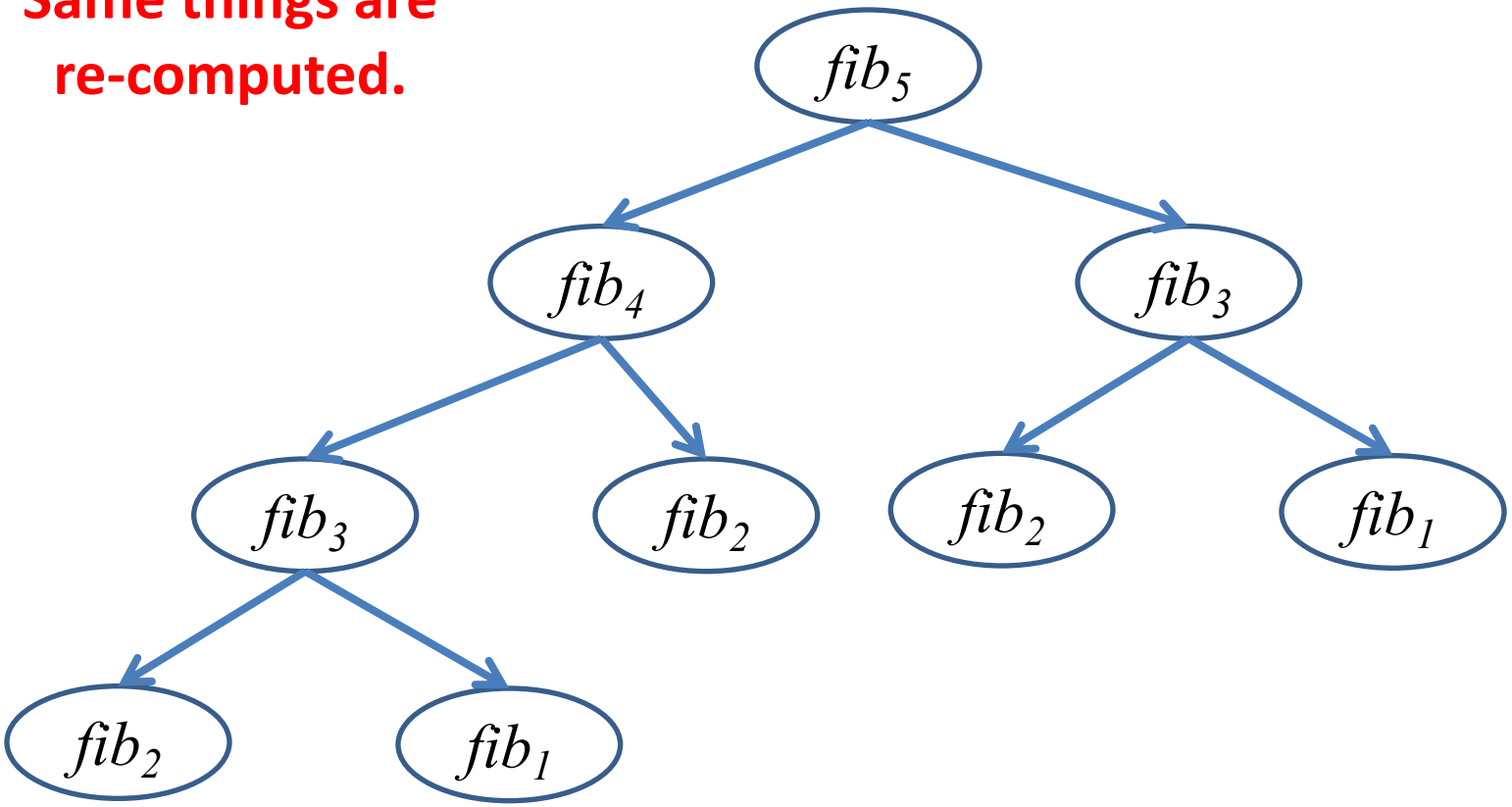
- Example: fibonacci numbers

$$fib_{1,2} = 1$$
$$fib_n = fib_{n-1} + fib_{n-2} \quad \ni \quad n > 2$$

```
define fibonacci(n)
  if n < 3 then
    return 1
  else
    return fibonacci(n − 1) + fibonacci(n − 2)
```

So, what is the problem with the recursive definition?

Same things are re-computed.



Alternatives to the naïve version of recursive fibonacci - 1

- Store intermediate results:

```
1  def fib(n):
2      results = [-1]*(n+1)
3      results[0] = 0
4      results[1] = 1
5      return recursive_fib(results, n)
6
7  def recursive_fib(results, n):
8      if results[n] < 0:
9          results[n] = recursive_fib(results, n-1)+recursive_fib(results,n-2)
10     else:
11         print "using previous result"
12     return results[n]
```

```
>>> fib(6)
using previous result
using previous result
using previous result
using previous result
using previous result
using previous result
```

Alternatives to the naïve version of recursive fibonacci - 2

- Go bottom to top:
  - Accumulate values on the way

```
1  def fib(n):
2      if n == 0:
3          return n
4      else:
5          return recursive_fib(n, 0, 0, 1)
6
7  def recursive_fib(n, i, f0, f1):
8      if n == i:
9          return f1
10     else:
11         return recursive_fib(n, i+1, f1, f0+f1)
```

Other times to avoid recursion

- When you have a limit on the memory
- When you have a limit on time
- When “divide & conquer” is not trivial/straightforward.

Consider these two implementations:

```
def fact1(n):
    if n == 0:
        return 1
    else:
        return n * fact1(n-1)
```

- The second implementation uses "**tail recursion**".
- tail recursion → the result of the **called function** is not used by the **calling function.**

```
def fact2(n):
    fact_helper(n, 1)

def fact_helper(n, r):
    if n == 0:
        return r
    else:
        return fact_helper(n-1, n*r)
```

Tail recursion & iteration

```
def fact2(n):
    fact_helper(n, 1)

def fact_helper(n, r):
    if n == 0:
        return r
    else:
        return fact_helper(n-1, n*r)
```

- Then, we can implement the tail-recursion version like on the right.

```
while n != 0
    r = n * r
    n = n - 1
```

Iteration

- More properly,

```
r = 1
while n != 0
    r = n * r
    n = n - 1
```

```
def fact2(n):
    fact_helper(n, 1)

def fact_helper(n, r):
    if n == 0:
        return r
    else:
        return fact_helper(n-1, n*r)
```

Iteration in Python

- while statement

```
1 while <condition> :
2         <statements>
```

- Example:

```
1 L = [2, 4, -10, "c"]
2 i = 0
3 while i < len(L) :
4         print L[i], "@"
5         i += 1
```

```
2 @
4 @
-10 @
c @
```

Iteration in Python

- for statement:

```
1 for <var> in <list> :
2         <statements>
```

- Example:

```
1 for x in [2, 4, -10, "c"]:
2         print x, "@"
```

```
2 @
4 @
-10 @
c @
```

Examples for Iteration

- Searching an item in a list

```
def is_member(Item, List):
    for x in List:
        if Item == x:
            return True
    return False
```

vs.

```
def is_member(Item, List):
    length = len(List)
    i = 0
    while i < length:
        if Item == List[i]:
            return i
    return -1
```

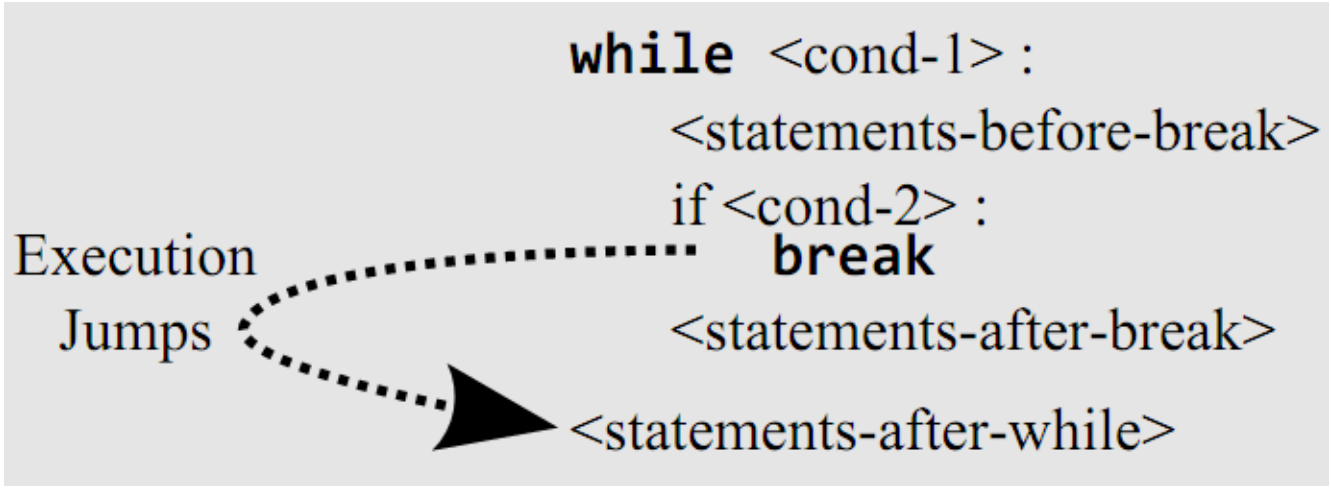
Nested Loops in Python

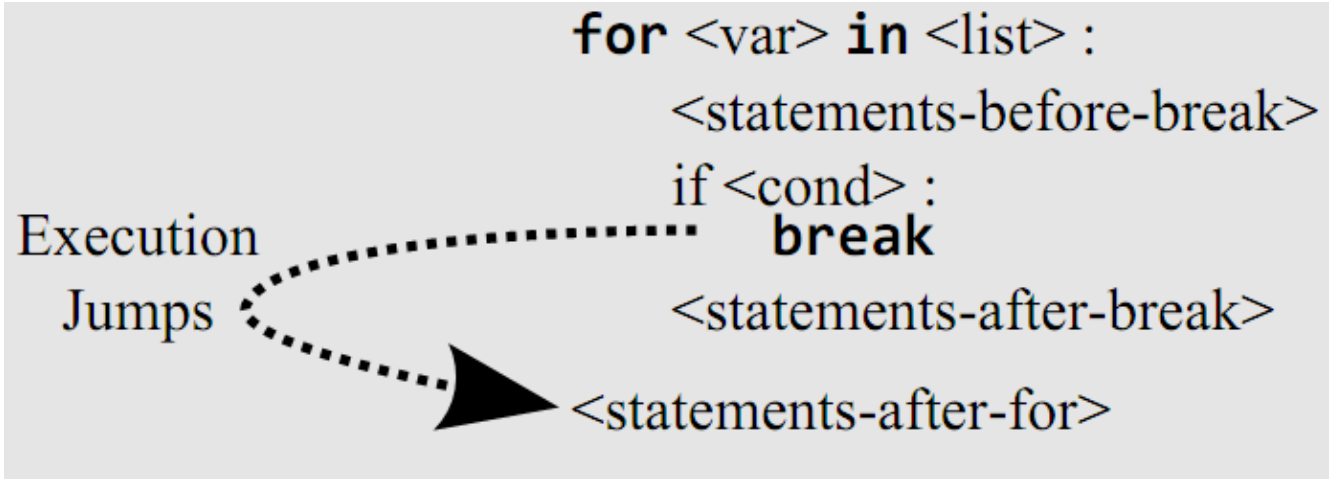
- You can put one loop within another one
  - No limit on nesting level

```
1 for i in range(1,10):
2         print i, ":",
3         for j in range(1,i):
4                 print j, "-",
5         print ""
```

```
1 :
2 : 1 -
3 : 1 - 2 -
4 : 1 - 2 - 3 -
5 : 1 - 2 - 3 - 4 -
6 : 1 - 2 - 3 - 4 - 5 -
7 : 1 - 2 - 3 - 4 - 5 - 6 -
8 : 1 - 2 - 3 - 4 - 5 - 6 - 7 -
9 : 1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 -
```

Break statements

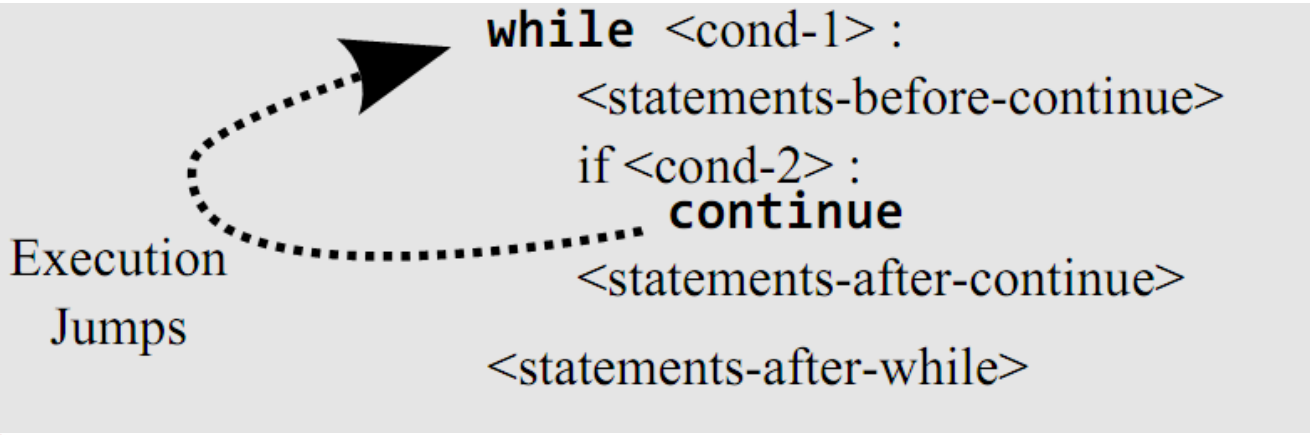


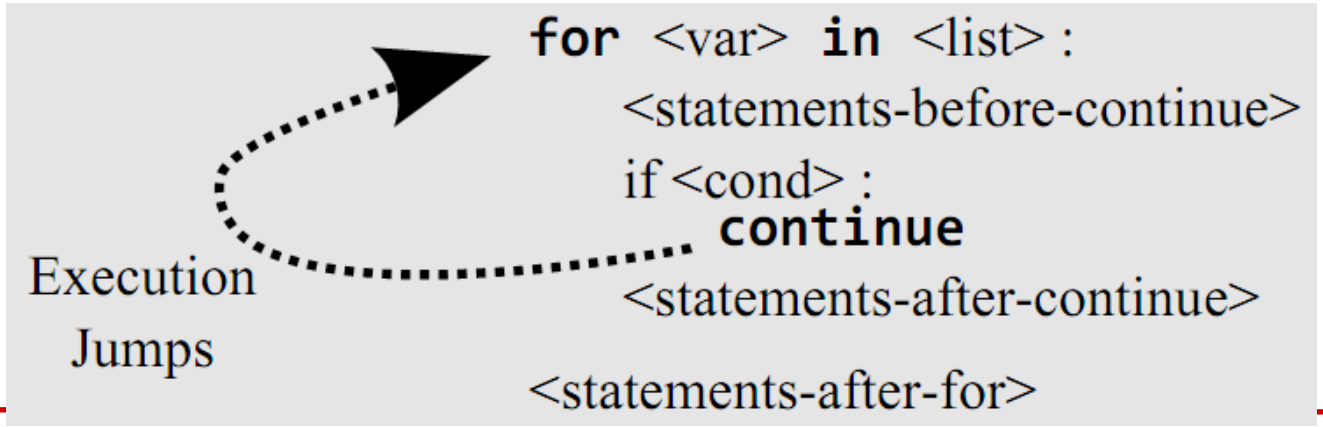


“break” example

```
1 x = 4
2 List = [1, 4, -2, 3, 8]
3 for m in List:
4         print m
5         if m == x:
6                 print "I have found a match"
7                 break
```

Continue statements





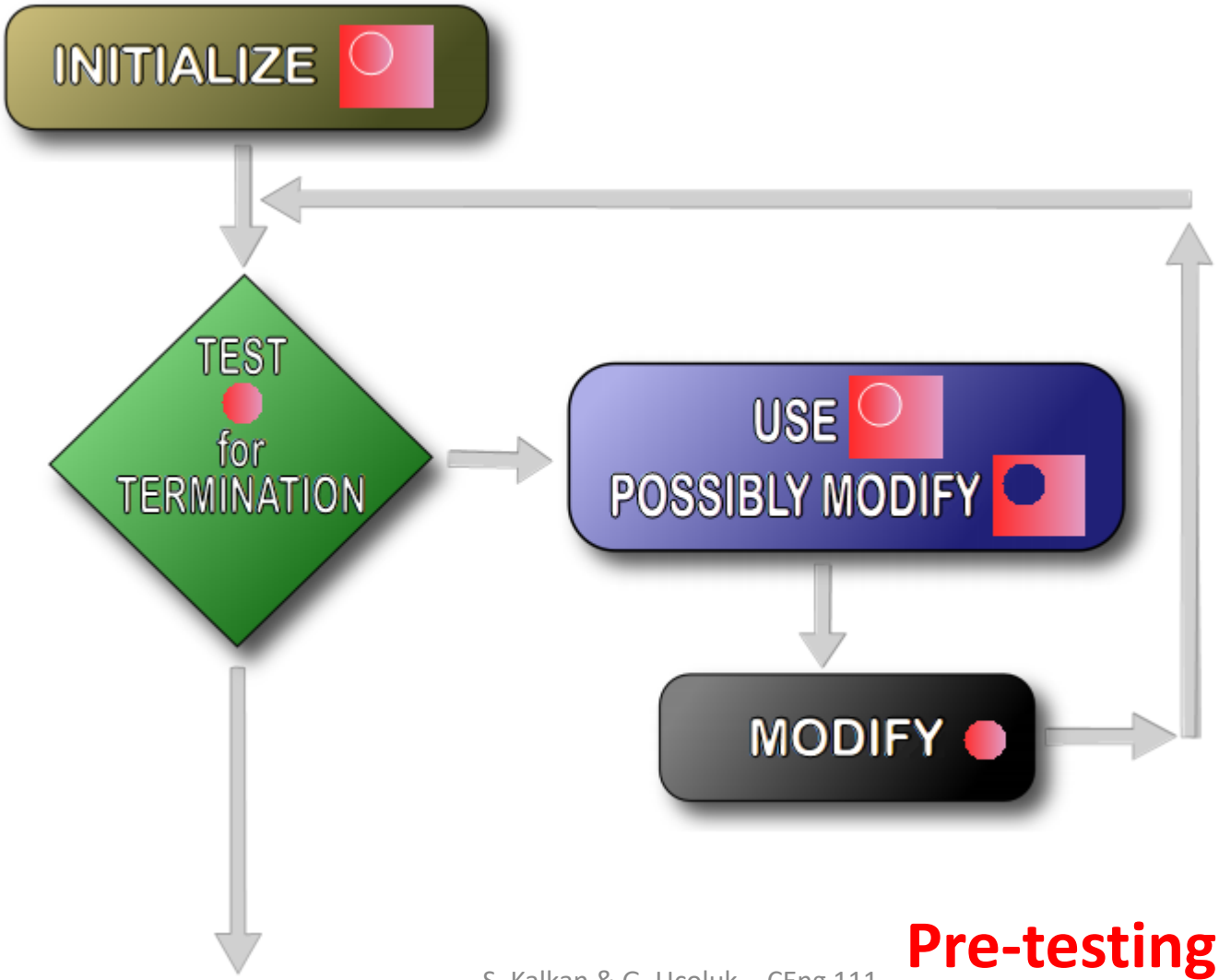
- <var> will point to the next item in the list.

Loops with “else:” parts

- The “else:” part is executed when the loop exits.
- If you use a “break” statement, the “else” part is not executed.

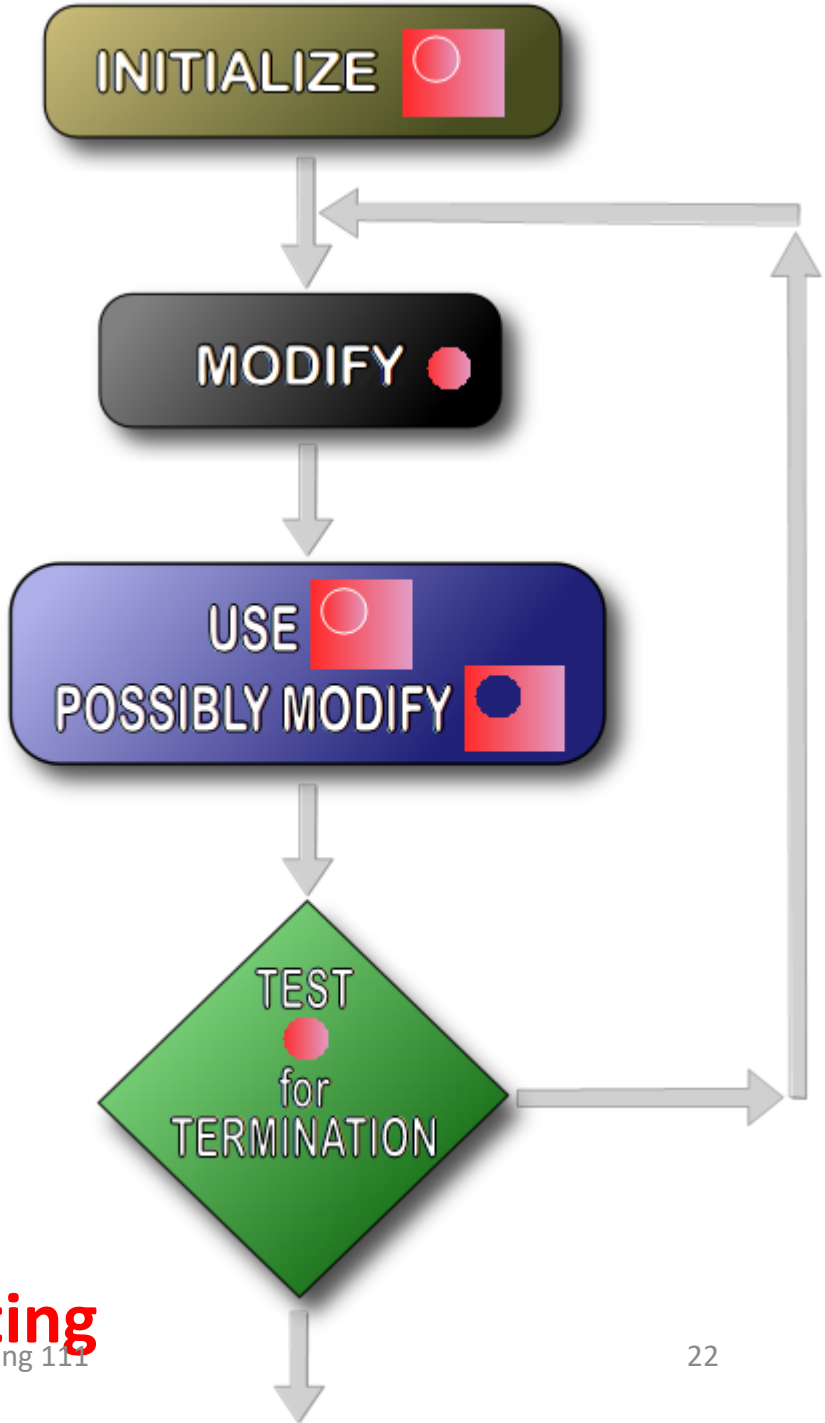
```
1 while <cond>:
2         <statements>
3 else:
4         <else-statements>
```

Types of Iterations



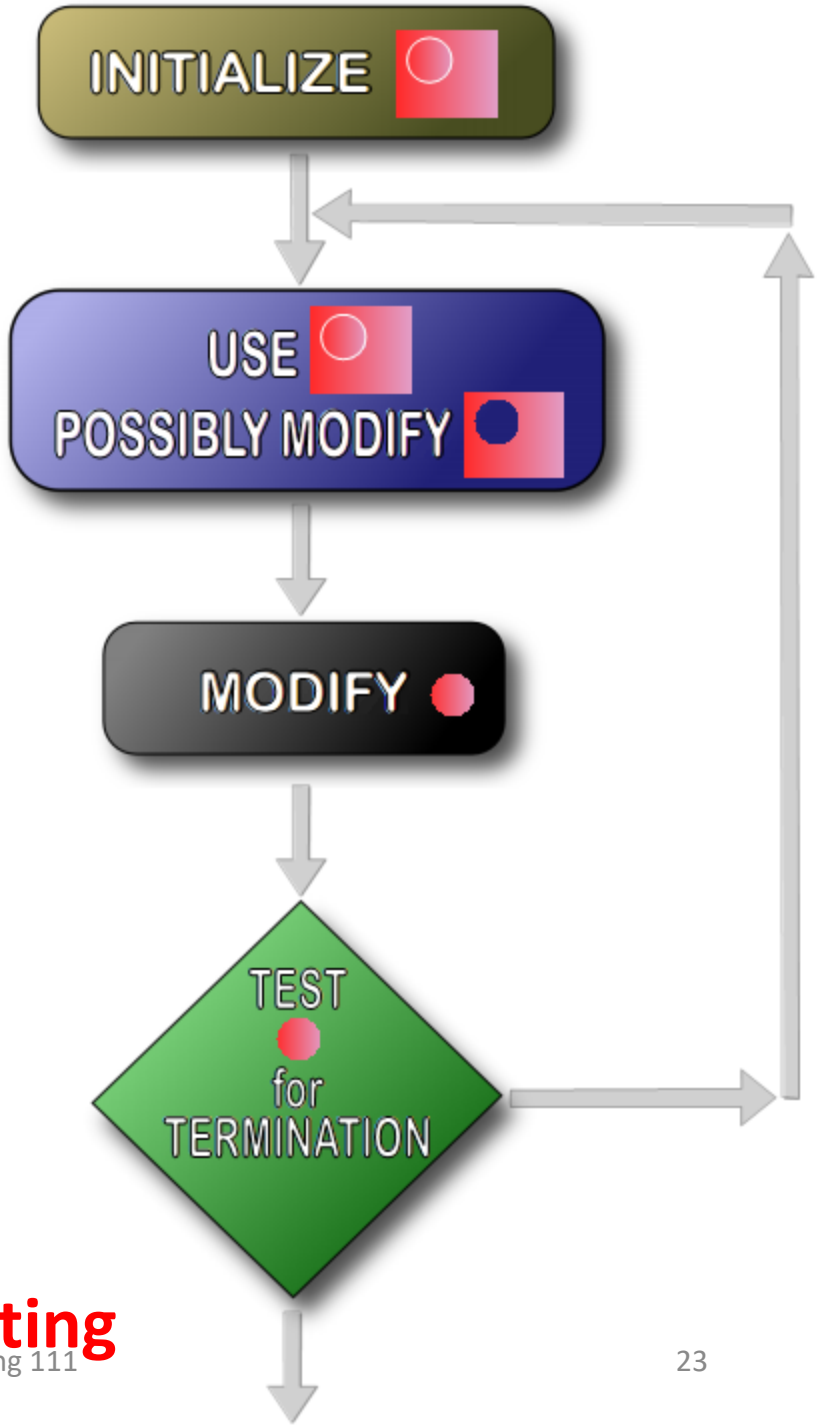
Pre-testing

Types of Iterations



Premod-posttesting

Types of Iterations



Postmod-posttesting

Examples for Iteration

- What does the following do?

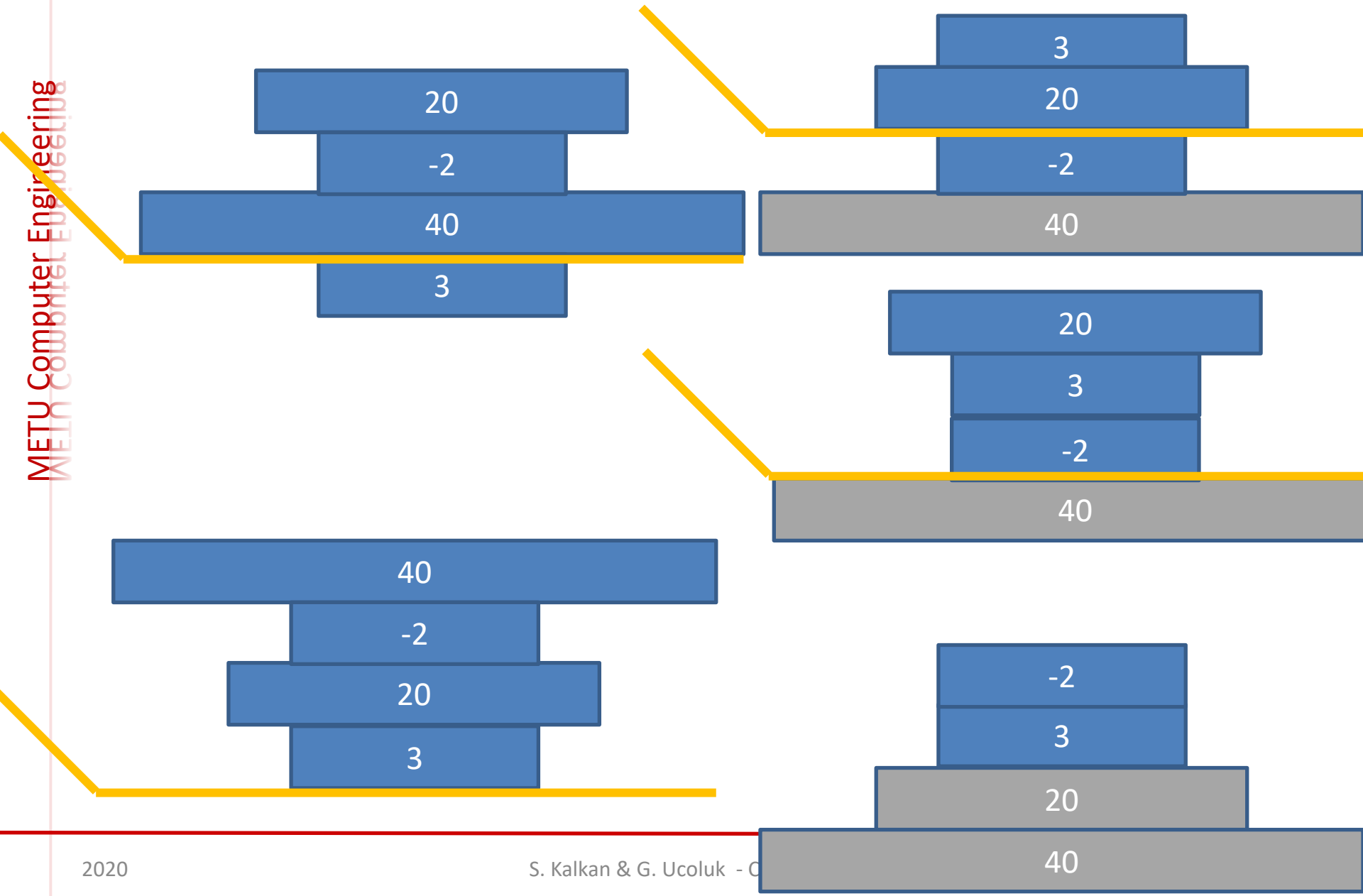
```
def f(List):
    length = len(List)
    changed = True
    while changed:
        changed = False
        i = 0
        while i < length-1:
            if List[i] > List[i+1]:
                (List[i], List[i+1]) = (List[i+1], List[i])
                changed = True
            i += 1
    return List
```

Another Example for Iteration

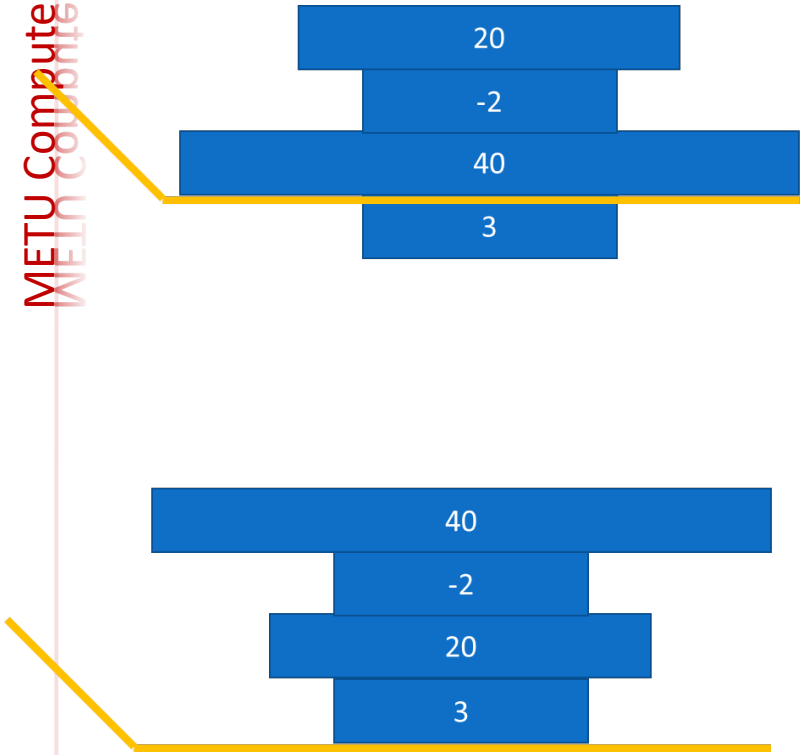
- Naïve selection sort

```
def select_min(L):
# Find, remove and return min
    Index = 0
    Min = L[Index]
    for i, x in enumerate(L):
        if x < Min:
            Index = i
            Min = x

    L.pop(Index)
    return Min

def naive_selection_sort(L):
    Result = [0]*len(L)
    for i in range(0, len(L)):
        x = select_min(L)
        Result[i] = x

    return Result
```

More examples for iteration: Pancake Sort



More examples for iteration: Pancake Sort




```
def find_max(L, s, e):
    maks = L[s]
    maks_ind = s
    for i in range(s, e):
        if L[i] > maks:
            maks_ind = i
            maks = L[i]
    return maks_ind
```

The lowest pancake is at the end of the list

```
def pancake_sort(L):
    N = len(L)
    for i in range(0, N):
        max_ind = find_max(L, 0, N-i)
        L[:max_ind+1] = L[max_ind::-1]
        L[:N-i] = L[N-i-1::-1]
    return L
```