More About Loops and Formatting

CS 8: Introduction to Computer Science
Lecture #13

Ziad Matni
Dept. of Computer Science, UCSB

Administrative

3 NEW ASSIGNMENTS!

Homework assignment #7 is due next Thursday (6/1)

Lab assignment #6 is due on Friday (5/26)

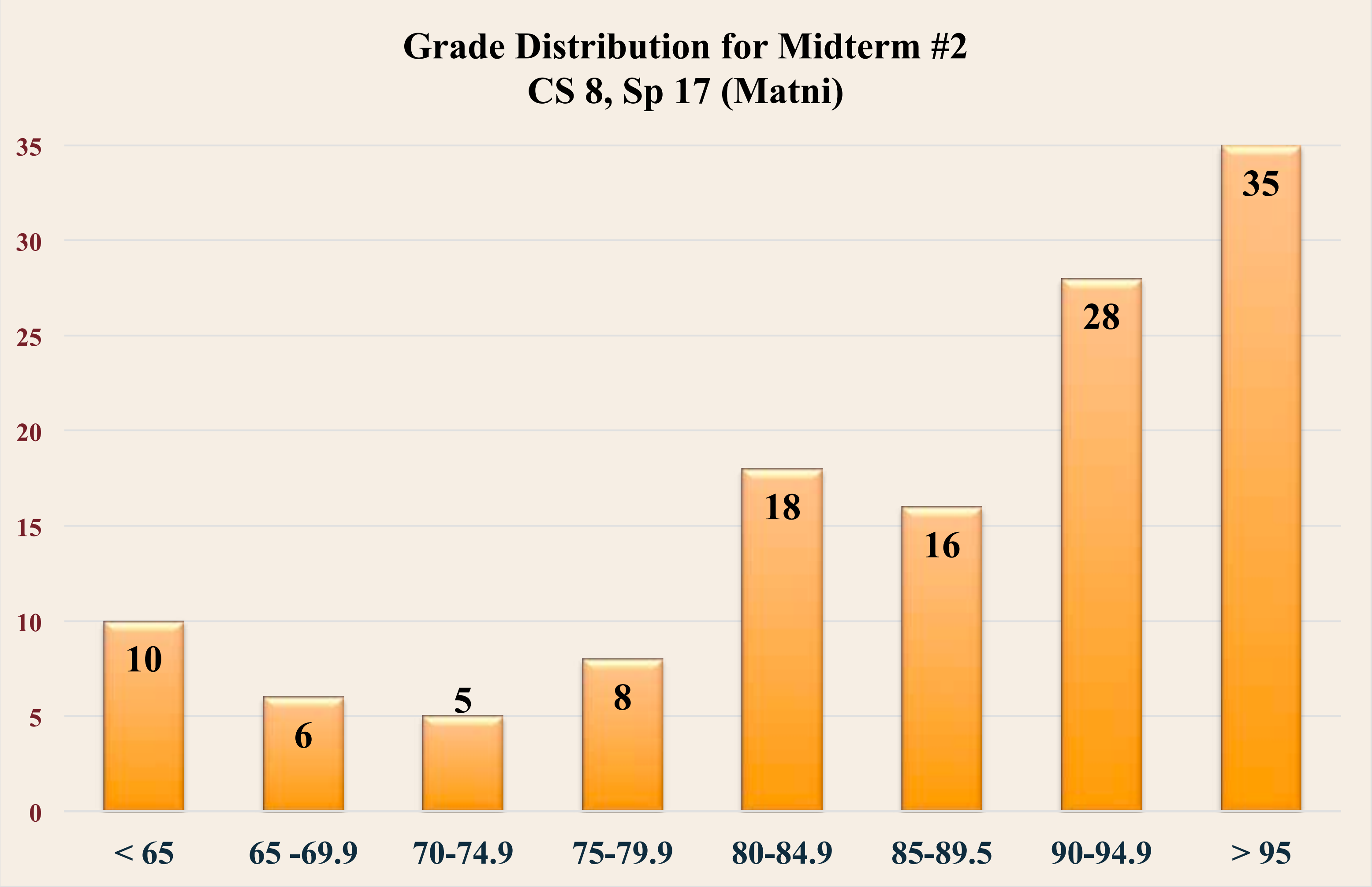
Project #2 is due on Tuesday, 6/6

Midterm #2 grades are now available!

A note about my grading choices

Midterm #2

Average = 85.8 Median = 89



Midterm#2 Questions

Consider the function below. What happens when it is called as tree(“t/o/p”)?

```
def tree(a):
        a.split(“/”)
        print (a[0] + 1)
```

A. You get an error message because you cannot perform arithmetic on strings
B. You get an error message because split is a reserved word.
C. It will print out “t1“
D. It will print out “top1”
E. It will print out “1op”

Midterm#2 Questions

What is the outcome of this Python code? You are given that chr(65) = ‘A’.

```
for i in range(1, 7):
    D[chr(65 + i)] = i
v = D['C'] * 3 - D['D'] * 2 + D['G']
D[‘Z’] = v

for x in list(D.keys()):
    if D[x] == v:
        print(x)
print (D[‘Z’])
```

Midterm#2 Questions

Write Python function, **Tripler**, takes in as parameter any numerical list, **alist**, as input parameter and returns a list with all the numbers in **alist** tripled.

For example:

if alist = [3, 2, 5], then **Tripler(alist) = [9, 6, 15]**, and

if alist = [-3, 10, 1, 7], then **Tripler(alist) = [-9, 30, 3, 21]**,

and so on.

Midterm#2 Questions

Consider the following Python function, makeD1:

```
def makeD1(myList):
    tempDict = {}
    for name in myList:
        tempDict[name] = myList.index(name)
    return(tempDict)
```

If you issue the Python statement:
myDict = makeD1(['bob', 'alice', 'bob']),
then what would the output of:

a) len(myDict)

b) myDict['bob']

c) myDict['alice']

Now consider this other Python function, makeD2:

```
def makeD2(myList):
    tempDict = {}
    for i in range( len(myList) ):
        tempDict[ myList[i] ] = i
    return(tempDict)
```

If you issue the Python statement:
myDict = makeD2(['bob', 'alice', 'bob']),
then what would the output of:

d) len(myDict)

e) myDict['bob']

f) myDict['alice']

Project #2

- Single program to write
  - But with many facets

- Program should:
  a) Ask the user for a file to read
  b) User can enter either a file name OR a URL
     - ➢ Program has to be able to detect if it's a URL
  c) Calculate how often all the characters in the file occur (frequency count)
  d) Print them out on the display in a pre-determined format
     - ➢ List the characters in ASCII code order
     - ➢ Some of these formats are done in a way that I'll explain in class later on

Repetition with a `while` loop

- `while` *condition*`:`

    `#` *executes over and over until false condition*

- Used for indefinite iteration
  - When it isn't possible to predict how many times a loop needs to execute
    - Unlike with for loops

- We use **`for`** loops for definite iteration (e.g., the loop executes exactly **n** times)

Repetition with a `while` loop

- While loops won't run at all if condition starts false
- While loops runs forever if condition stays true
- Sometimes helps to use `break` to exit loop, or `continue` to restart loop (these work with `for` loops too)
  - But we don't like to use break/continue too much
  - Makes for sloppy algorithms

Applying `while`

- Can be used for counter-controlled loops:

```
n = 500
counter = 0                        #  (1) initialize
while counter < n:                 #  (2) check condition
    print(counter * counter)
    counter = counter + 1          #  (3) change state
```

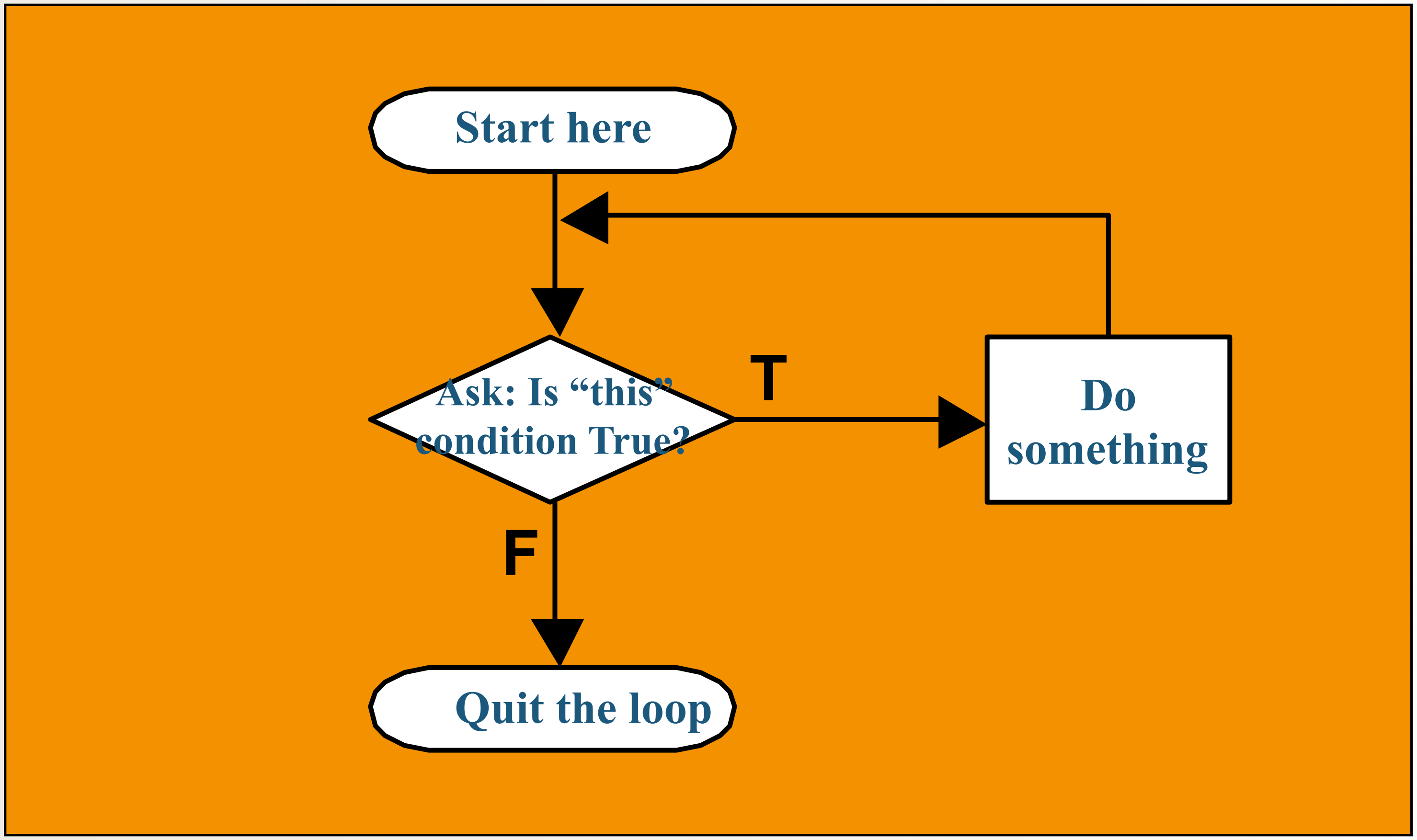
  - But this is a definite loop – easier to use `for`

Applying while

- Better application – unlimited data entry:

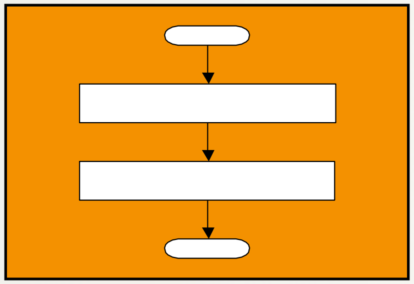
```
# (1) initialize
AllGrades = 0
grade = input("enter grade or q to quit: ")
# (2) check condition
while grade != "q":
    # process grade here, then get next one
    AllGrades = AllGrades + int(grades)
    # (3) change states
    grade = input("enter grade or q to quit: ")
# While loop has ended, now do other stuff…
```

Flow of an Iteration Structure

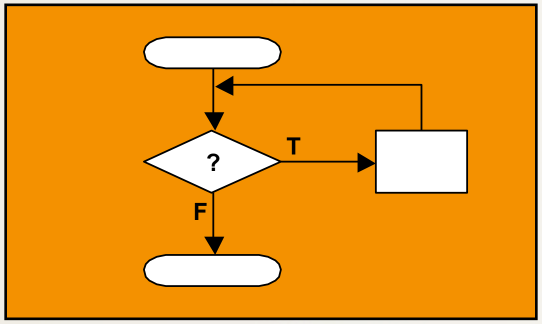


Review: 3 Control Structure Types

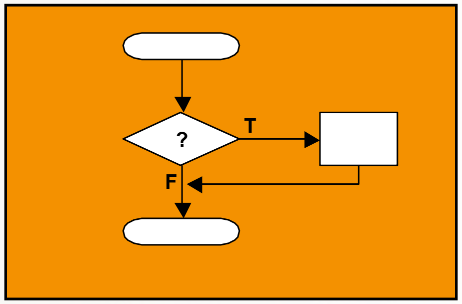
Sequence

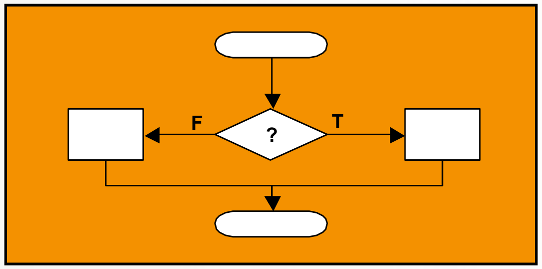


Iteration

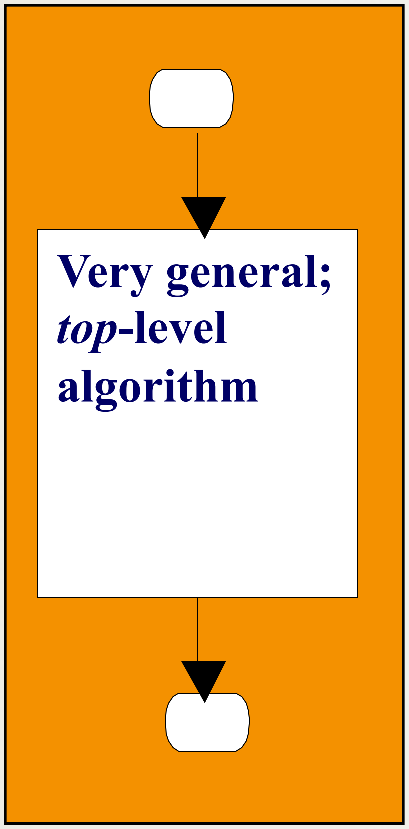


Selection





Structure "Rule" #1:
start with the simplest flowchart

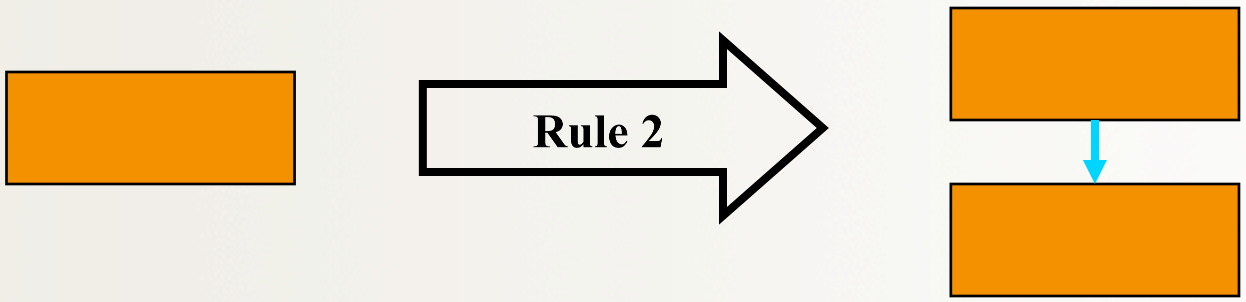


- Really just a way to start; clarifies the "big picture"
- For example:

  get some data, calculate and then show some results
- Notice: just one rectangle

Rule #2:

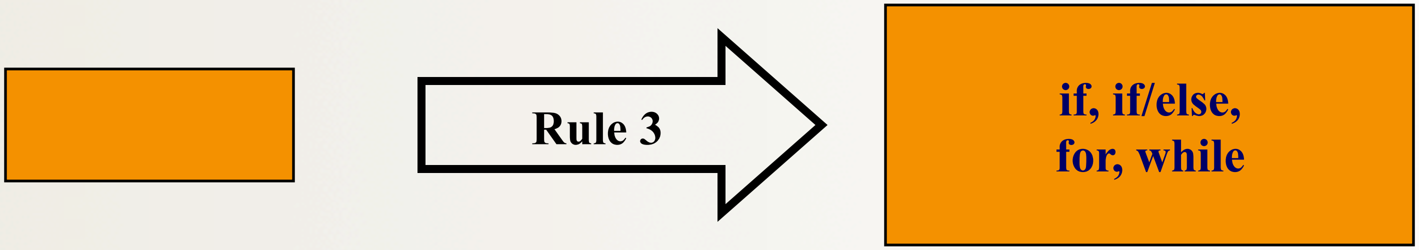
replace any rectangle by two rectangles in sequence



- This “stacking rule” can apply repeatedly
- For example:

1. Get data
2. Process
3. Show results

Rule #3: replace any rectangle by any control structure



- This "nesting rule" also applies repeatedly – each control structure has its own rectangles
- e.g., nest a `while` loop in an `if` structure:

```
if  n > 0:
   while i < n:
      print(i)
      i = i + 1
```

Rule #4: apply rules #2 and #3 repeatedly, and in any order

- Stack, nest, stack, nest, nest, stack, … gets more and more detailed as one proceeds
  - Think of control structures as building blocks that can be *combined in two ways only*.
- Overall process is known as "top-down design by stepwise refinement"
- Fact: *any algorithm* can be written as a combination of sequence, selection, and iteration structures.

</LECTURE>