Exposure CS 2021 for CS1

Chapter 8 Section 1-5 Slides

Repetition Control Structures & Using Them with Turtle Graphics

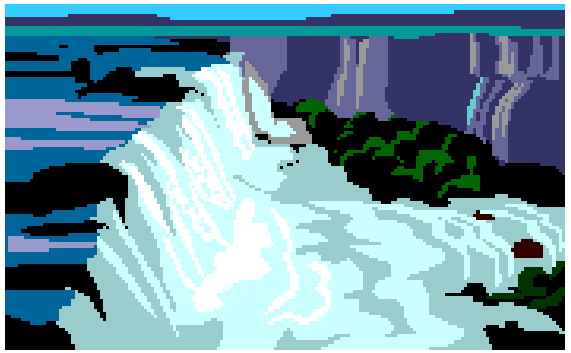
PowerPoint Presentation
created by:
Mr. John L. M. Schram
and Mr. Leon Schram
Authors of Exposure
Computer Science

Section 8.1

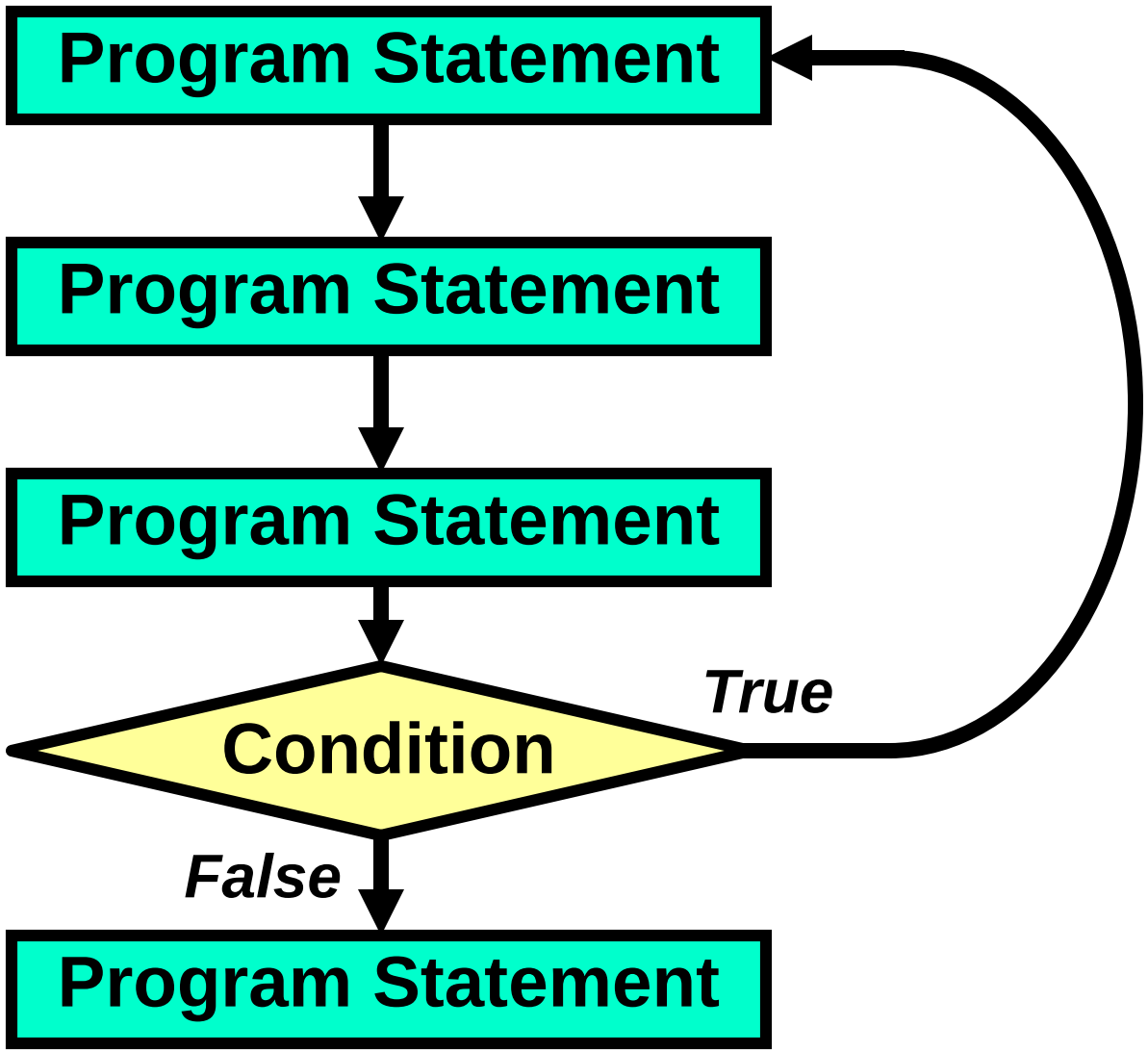
Introduction

Program Flow Review

Program Flow **follows the exact sequence of listed program statements, unless directed otherwise by a Python control structure.**



Repetition Review



Section 8.2

Fixed Repetition

```
 1 # Repetition01.py
 2 # This program displays 20 identical lines of text.
 3 # The program is very inefficient in that it uses
 4 # 20 separate <print> statements.
 5
 6
 7 print()
 8 print("Eat at Joe's Friendly Diner for the best lunch value!")
 9 print("Eat at Joe's Friendly Diner for the best lunch value!")
10 print("Eat at Joe's Friendly Diner for the best lunch value!")
11 print("Eat at Joe's Friendly Diner for the best lunch value!")
12 print("Eat at Joe's Friendly Diner for the best lunch value!")
13 print("Eat at Joe's Friendly Diner for the best lunch value!")
14 print("Eat at Joe's Friendly Diner for the best lunch value!")
15 print("Eat at Joe's Friendly Diner for the best lunch value!")
16 print("Eat at Joe's Friendly Diner for the best lunch value!")
17 print("Eat at Joe's Friendly Diner for the best lunch value!")
18 print("Eat at Joe's Friendly Diner for the best lunch value!")
19 print("Eat at Joe's Friendly Diner for the best lunch value!")
20 print("Eat at Joe's Friendly Diner for the best lunch value!")
21 print("Eat at Joe's Friendly Diner for the best lunch value!")
22 print("Eat at Joe's Friendly Diner for the best lunch value!")
23 print("Eat at Joe's Friendly Diner for the best lunch value!")
24 print("Eat at Joe's Friendly Diner for the best lunch value!")
25 print("Eat at Joe's Friendly Diner for the best lunch value!")
26 print("Eat at Joe's Friendly Diner for the best lunch value!")
27 print("Eat at Joe's Friendly Diner for the best lunch value!")
```

```
 1 # Repetitio
 2 # This prog
 3 # The progr
 4 # 20 separa
 5
 6
 7 print()
 8 print("Eat
 9 print("Eat
10 print("Eat
11 print("Eat
12 print("Eat
13 print("Eat
14 print("Eat
15 print("Eat
16 print("Eat
17 print("Eat
18 print("Eat
19 print("Eat
20 print("Eat
21 print("Eat
22 print("Eat
23 print("Eat
24 print("Eat
25 print("Eat
26 print("Eat
27 print("Eat
```

```
----jGRASP exec: python Repetition01.py

Eat at Joe's Friendly Diner for the best lunch value!
Eat at Joe's Friendly Diner for the best lunch value!
Eat at Joe's Friendly Diner for the best lunch value!
Eat at Joe's Friendly Diner for the best lunch value!
Eat at Joe's Friendly Diner for the best lunch value!
Eat at Joe's Friendly Diner for the best lunch value!
Eat at Joe's Friendly Diner for the best lunch value!
Eat at Joe's Friendly Diner for the best lunch value!
Eat at Joe's Friendly Diner for the best lunch value!
Eat at Joe's Friendly Diner for the best lunch value!
Eat at Joe's Friendly Diner for the best lunch value!
Eat at Joe's Friendly Diner for the best lunch value!
Eat at Joe's Friendly Diner for the best lunch value!
Eat at Joe's Friendly Diner for the best lunch value!
Eat at Joe's Friendly Diner for the best lunch value!
Eat at Joe's Friendly Diner for the best lunch value!
Eat at Joe's Friendly Diner for the best lunch value!
Eat at Joe's Friendly Diner for the best lunch value!

----jGRASP: operation complete.
```

```
1 # Repetition02.py
2 # This program displays 20 identical lines of text like
3 # the last program, but is much more efficient because
4 # is uses a <for> loop.
5
6
7 print()
8
9 for k in range(20):
10   print("Eat at Joe's Friendly Diner for the best lunch value!")
11
```

The **range** value indicates how many times the **for** loop will repeat.

k is the *loop counter* or LCV (**L***oop* **C***ontrol* **V***ariable*).

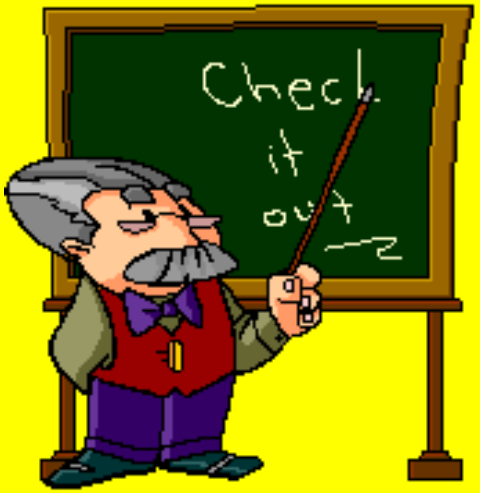
It is considered “OK” to use a single letter variable for a loop counter. This is one of the very few times that it is considered “OK” to use a single letter variable.

```
 1 # Repetition03.py
 2 # This program demonstrates the Syntax Error
 3 # you receive when you do not properly indent
 4 # the programming statement(s) being controlled
 5 # by a control structure.
 6
 7 # NOTE: In most languages, indentation is recommended.
 8 #       In Python, indentation is required.
 9
10
11 for k in range(20):
12 print("Eat at Joe's Friendly Diner for the best lunch value!")
```

```
----jGRASP exec: python Repetition03.py
 File "Repetition03.py", line 12
   print("Eat at Joe's Friendly Diner for the
best lunch value!")
          ^
IndentationError: expected an indented block

----jGRASP wedge2: exit code for process is 1.
----jGRASP: operation complete.
```

Indentation Rule Review:

In most languages, indenting the program statements that are "controlled" by control structures is recommended.



In Python, it is required.

Python programs that do not use proper and consistent indentation will not execute.

```
1  # Repetition04.py
2  # This program displays the value of the loop counter
3  # which is also called the "Loop Control Variable" (LCV).
4  # Note that even though the loop repeats 20 times,
5  # the counter actually counts from 0 to 19.
6
7
8  print()
9
10 for k in range(20):
11     print(k, end = " ")
12
13 print()
```

```
 1 # Repetition04.py
 2 # This program displays the value of the loop counter
 3 # which is also called the "Loop Control Variable" (LCV).
 4 # Note that even though the loop repeats 20 times,
 5 # the counter actually counts from 0 to 19.
 6
 7
 8 print()
 9
10 for k in range(20):
11     print(k, end = " ")
12
13 print()
```

```
  ----jGRASP exec: python Repetition04.py

 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

  ----jGRASP: operation complete.
```

```
1  # Repetition05.py
2  # This program demonstrates that multiple program
3  # statements can be controlled with a <for>
4  # loop structure as long as proper,
5  # consistent indentation is used.
6
7
8  print()
9
10 for k in range(10):
11     print("############################")
12     print("##    Box Number",k,"   ##")
13     print("############################")
14     print()
15
```

```
----jGRASP exec: python Repetition05.py

######################
##   Box Number 0   ##
######################

######################
##   Box Number 1   ##
######################

######################
##   Box Number 2   ##
######################

######################
##   Box Number 3   ##
######################

######################
##   Box Number 4   ##
######################

######################
##   Box Number 5   ##
######################

######################
##   Box Number 6   ##
######################

######################
##   Box Number 7   ##
######################

######################
##   Box Number 8   ##
######################

######################
##   Box Number 9   ##
######################


----jGRASP: operation complete.
```

```
1  # Repetition06.py
2  # This program demonstrates how to make the <for>
3  # loop start counting at a number other than zero.
4  # The secret is to use 2 numbers in the <range> command.
5  # The counter will begin with the first number.
6  # and stop before it reaches the second number.
7
8
9  print()
10
11 for k in range(10,30): # Displays 10 to 29
12    print(k, end = " ")
13
14 print("\n")
15
16 for k in range(10,31): # Displays 10 to 30
17    print(k, end = " ")
18
19 print()
```

```
----jGRASP exec: python Repetition06.py

10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29

10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

----jGRASP: operation complete.
```

```
 9 print()
10
11 for k in range(10,30): # Displays 10 to 29
12     print(k, end = " ")
13
14 print("\n")
15
16 for k in range(10,31): # Displays 10 to 30
17     print(k, end = " ")
18
19 print()
```

```
 1 # Repetition07.py
 2 # This program demonstrates how to change the "step"
 3 # value in the <for> loop.  By default it is 1.
 4 # To count by a number other than 1 requires adding
 5 # a third number to the <range> command.
 6 # NOTE: As before, you may need to add 1 to the
 7 # "stopping value" to make the loop work properly.
 8
 9
10 print()
11
12 for k in range(10,30,2): # Displays evens from 10-28
13     print(k, end = " ")
14
15 print("\n")
16
17 for k in range(10,31,2): # Displays evens from 10-30
18     print(k, end = " ")
19
20 print("\n")
21
```

```
22 for k in range(5,101,5): # Displays 5 to 100 by 5s
23     print(k, end = " ")
24
25 print("\n")
26
27 for k in range(50,81,3): # Displays 50 to 80 by 3s
28     print(k, end = " ")
29
30 print("\n")
31
32 for k in range(20,0,-1): # Displays 20 down to 1
33     print(k, end = " ")
34
35 print("\n")
36
37
38 for k in range(20,-1,-1): # Displays 20 down to 0
39     print(k, end = " ")
40
41 print()
42
```

```
  ----jGRASP exec: python Repetition07.py

10 12 14 16 18 20 22 24 26 28

10 12 14 16 18 20 22 24 26 28 30

5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100

50 53 56 59 62 65 68 71 74 77 80

20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

  ----jGRASP: operation complete.
```

Fixed Repetition

Python has two control structures for repetition.

Other computer science terms for *repetition* are *looping* and *iteration*.

Fixed Repetition is done with the **for** loop structure.

General Syntax:

```
for LCV in range(Start,Stop,Step):
    execute program statement(s)
```

The *LCV* is the *Loop Control Variable* or *Loop Counter*.

Start specifies the first counting value.
If not specified, the default value is **0**.

The loop will "stop" before it reaches the *Stop* value.

Step specifies what the loop "counts by".
If not specified, the default value is **1**.

Fixed Repetition (continued)

Specific Examples:

```
for k in range (10):          # Counts from 0 to 9
    print(k, end = " ")

for k in range (10,20):       # Counts from 10 to 19
    print(k, end = " ")

for k in range (10,21,2):     # Counts from 10 to 20 by 2s
    print(k, end = " ")
```

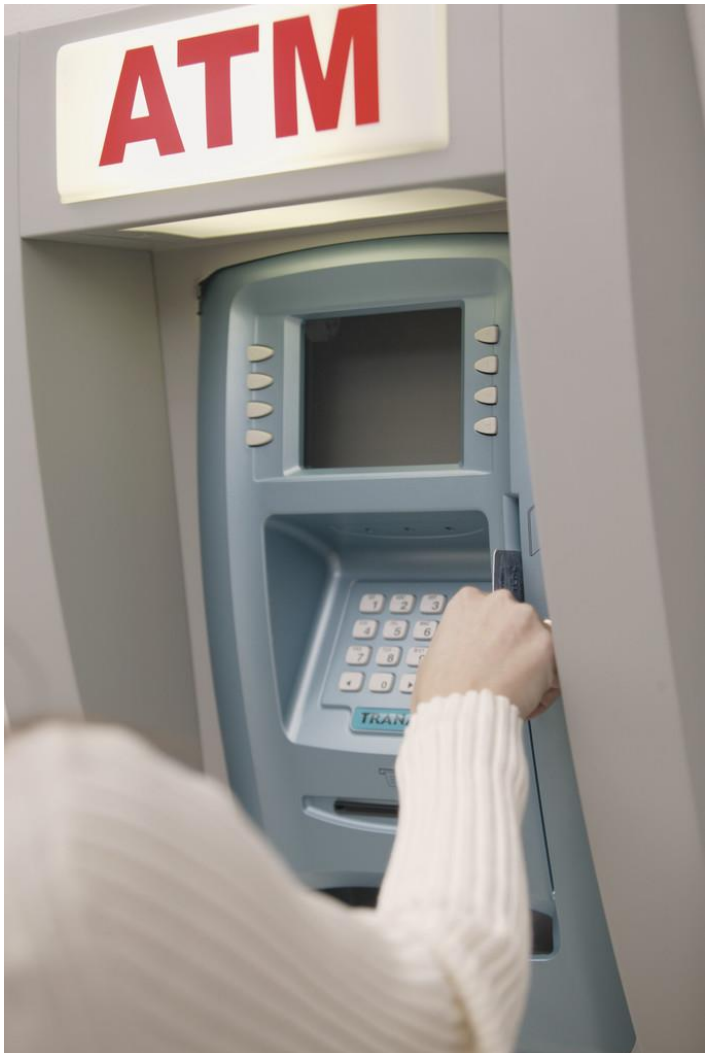
Repetition of multiple program statements works fine as long as proper, consistent indentation is used.

Section 8.3

Conditional Repetition

Conditional Repetition

Real Life Examples



```
1  # Repetition08.py
2  # This program is supposed to keep repeating until
3  # a correct PIN of 5678 is entered.
4  # The program does not work because the <for>
5  # loop is used at a time that is not appropriate.
6  # The <for> loop is meant for "fixed" repetition.
7  # Entering a PIN is an example of "conditional" repetition.
8
9
10 for k in range(10):
11     print()
12     pin = input("Enter 4 digit PIN#.  -->  ")
13     if pin != "5678":
14         print("\nThat is not the correct PIN.  Try Again.")
15
16 print("\nYou are now logged in.  Welcome to the program.")
17
```

```
1  # Repetition08.py
2  # This program is supposed
3  # a correct PIN of 5678 is
4  # The program does not wor
5  # loop is used at a time t
6  # The <for> loop is meant
7  # Entering a PIN is an exam
8
9
10 for k in range(10):
11     print()
12     pin = input("Enter 4 di
13     if pin != "5678":
14         print("\nThat is not
15
16 print("\nYou are now logged
17
```

```
----jGRASP exec: python Repetition08.py

Enter 4 digit PIN#.  -->  1234

That is not the correct PIN.  Try Again.

Enter 4 digit PIN#.  -->  2345

That is not the correct PIN.  Try Again.

Enter 4 digit PIN#.  -->  3456

That is not the correct PIN.  Try Again.

Enter 4 digit PIN#.  -->  4567

That is not the correct PIN.  Try Again.

Enter 4 digit PIN#.  -->  5678

Enter 4 digit PIN#.  -->  5678

Enter 4 digit PIN#.  -->  5678

Enter 4 digit PIN#.  -->  5678

Enter 4 digit PIN#.  -->  5678

Enter 4 digit PIN#.  -->  0000

That is not the correct PIN.  Try Again.

You are now logged in.  Welcome to the program.

----jGRASP: operation complete.
```

```
1  # Repetition09.py
2  # This program fixes the problem of the previous program
3  # by using a <while> loop.  Now the loop will stop when
4  # the correct PIN of 5678 is entered.
5
6
7  pin = ""
8  while pin != "5678":
9      print()
10     pin = input("Enter 4 digit PIN#.  -->  ")
11     if pin != "5678":
12         print("\nThat is not the correct PIN.  Try Again.")
13
14 print("\nYou are now logged in.  Welcome to the program.")
15
```

```
1  # Repetition09.py
2  # This program fixes the pr
3  # by using a <while> loop.
4  # the correct PIN of 5678 i
5
6
7  pin = ""
8  while pin != "5678":
9      print()
10     pin = input("Enter 4 dig
11     if pin != "5678":
12         print("\nThat is not
13
14 print("\nYou are now logged in.  Welcome to the program.")
15
```

```
----jGRASP exec: python Repetition09.py

Enter 4 digit PIN#.  -->  1234

That is not the correct PIN.  Try Again.

Enter 4 digit PIN#.  -->  2345

That is not the correct PIN.  Try Again.

Enter 4 digit PIN#.  -->  3456

That is not the correct PIN.  Try Again.

Enter 4 digit PIN#.  -->  4567

That is not the correct PIN.  Try Again.

Enter 4 digit PIN#.  -->  5678

You are now logged in.  Welcome to the program.

----jGRASP: operation complete.
```

Conditional Repetition

General Syntax:

initialize condition variable
while condition is True:
    execute program statement(s)

Specific Example:

```
password = ""
while password != "Qwerty2018":
    password = input("Enter password.  -->  ")
    if password != "Qwerty2018":
        print("Wrong password. Please re-enter")
print("Welcome.")
```

Fixed Repetition vs. Conditional Repetition

Fixed Repetition describes a situation where you know – ahead of time – how many times you want the loop to repeat.

An example would be drawing exactly 100 circles on the screen.

The command for fixed repetition is **for**.

Conditional Repetition describes a situation where you do NOT know how many times the loop will repeat.

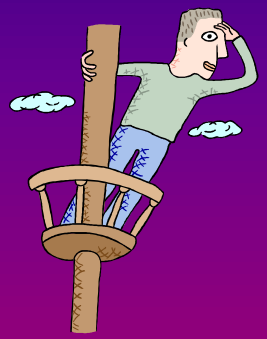
The loop has to repeat until some *condition* is met.

An example would be entering a password.

The command for conditional repetition is **while**.

Section 8.4

Nested Control Structures



```
1 # Nested01.java
2 # This program demonstrates "Nested Repetition"
3 # which is one type of "Nested Control Structure".
4 # Since the outer loop repeats 3 times and the
5 # inner loop repeats 4 times, the word "Hello"
6 # is displayed 3 * 4 or 12 times.
7
8
9 print()
10
11 for j in range(3):
12     for k in range(4):
13         print("Hello")
14
```

```
1 # Nested01.java
2 # This program demonstrates "Neste
3 # which is one type of "Nested Con
4 # Since the outer loop repeats 3 t
5 # inner loop repeats 4 times, the
6 # is displayed 3 * 4 or 12 times.
7
8
9 print()
10
11 for j in range(3):
12     for k in range(4):
13         print("Hello")
14
```

```
----jGRASP

Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello

----jGRASP:
```

```
1 # Nested02.java
2 # This program displays the value of the counters
3 # for both loops.  Note that the inner loop counts
4 # much faster than the outer loop.
5
6
7 print()
8
9 for j in range(3):
10     for k in range(4):
11         print(j,k)
12
```

```
1 # Nested02.java
2 # This program displays the value
3 # for both loops.  Note that the
4 # much faster than the outer loop
5
6
7 print()
8
9 for j in range(3):
10     for k in range(4):
11         print(j,k)
12
```

```
----jGRASP

0 0
0 1
0 2
0 3
1 0
1 1
1 2
1 3
2 0
2 1
2 2
2 3

----jGRASP:
```

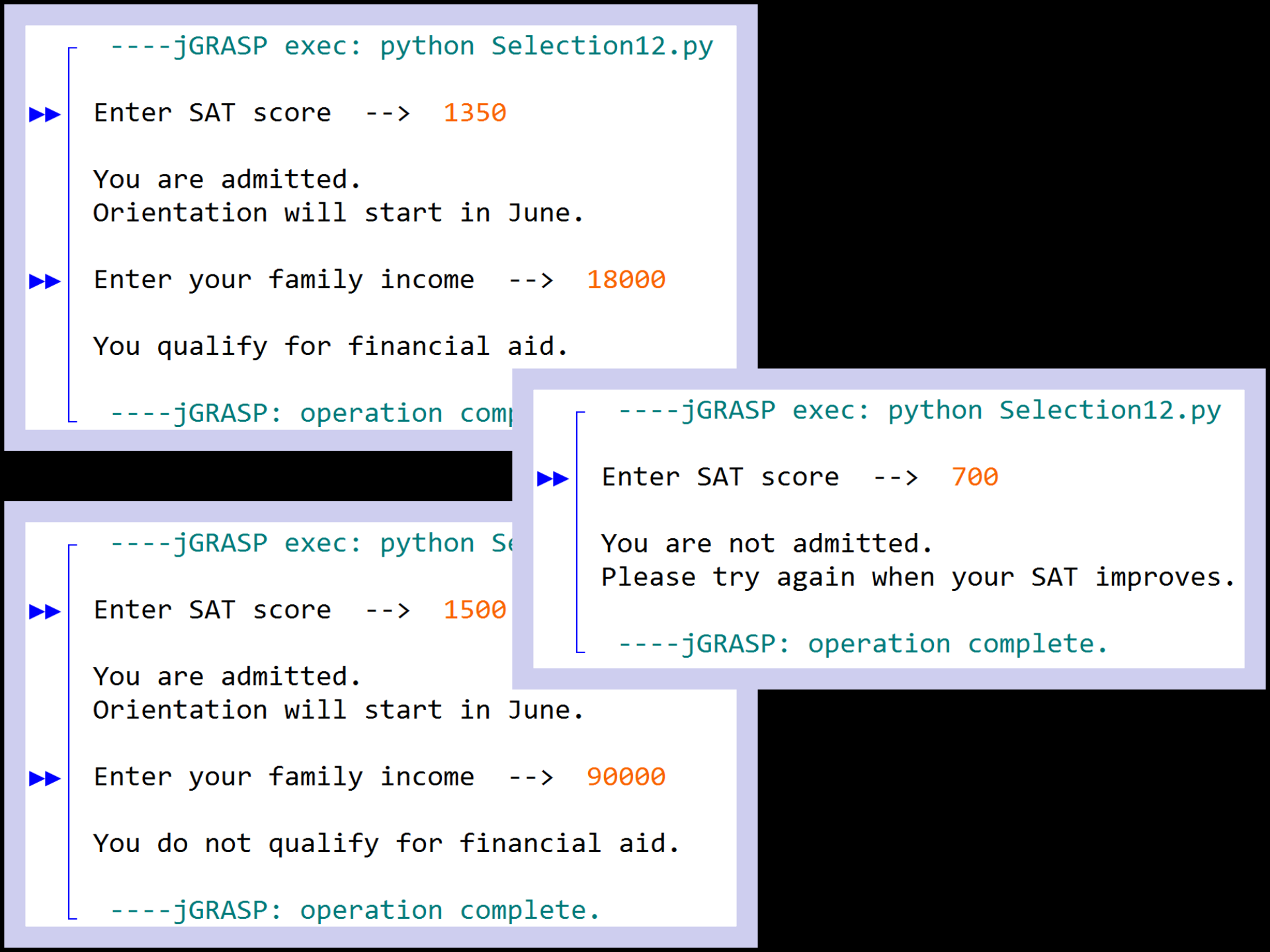
```
1  # Nested03.java
2  # This program displays a times table
3  # that goes from 1 * 1 to 15 * 15.
4  # In this program, the table does not
5  # line up properly.
6
7
8  print()
9
10 for r in range(1,16):
11     for c in range(1,16):
12         print(r * c, end = " ")
13     print()
14
```

```
 ----jGRASP exec: python Nested03.py

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
2 4 6 8 10 12 14 16 18 20 22 24 26 28 30
3 6 9 12 15 18 21 24 27 30 33 36 39 42 45
4 8 12 16 20 24 28 32 36 40 44 48 52 56 60
5 10 15 20 25 30 35 40 45 50 55 60 65 70 75
6 12 18 24 30 36 42 48 54 60 66 72 78 84 90
7 14 21 28 35 42 49 56 63 70 77 84 91 98 105
8 16 24 32 40 48 56 64 72 80 88 96 104 112 120
9 18 27 36 45 54 63 72 81 90 99 108 117 126 135
10 20 30 40 50 60 70 80 90 100 110 120 130 140 150
11 22 33 44 55 66 77 88 99 110 121 132 143 154 165
12 24 36 48 60 72 84 96 108 120 132 144 156 168 180
13 26 39 52 65 78 91 104 117 130 143 156 169 182 195
14 28 42 56 70 84 98 112 126 140 154 168 182 196 210
15 30 45 60 75 90 105 120 135 150 165 180 195 210 225

 ----jGRASP: operation complete.
```

```
1 # Nested04.java
2 # This program displays a better times table
3 # where everything lines up properly by using
4 # the <format> command.
5
6
7 print()
8
9 for r in range(1,16):
10     for c in range(1,16):
11         print("{:3}".format(r * c), end = " ")
12     print()
13
```

```
----jGRASP exec: python Nested04.py

 1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
 2   4   6   8  10  12  14  16  18  20  22  24  26  28  30
 3   6   9  12  15  18  21  24  27  30  33  36  39  42  45
 4   8  12  16  20  24  28  32  36  40  44  48  52  56  60
 5  10  15  20  25  30  35  40  45  50  55  60  65  70  75
 6  12  18  24  30  36  42  48  54  60  66  72  78  84  90
 7  14  21  28  35  42  49  56  63  70  77  84  91  98 105
 8  16  24  32  40  48  56  64  72  80  88  96 104 112 120
 9  18  27  36  45  54  63  72  81  90  99 108 117 126 135
10  20  30  40  50  60  70  80  90 100 110 120 130 140 150
11  22  33  44  55  66  77  88  99 110 121 132 143 154 165
12  24  36  48  60  72  84  96 108 120 132 144 156 168 180
13  26  39  52  65  78  91 104 117 130 143 156 169 182 195
14  28  42  56  70  84  98 112 126 140 154 168 182 196 210
15  30  45  60  75  90 105 120 135 150 165 180 195 210 225


----jGRASP: operation complete.
```

```
 1 # Nested05.py
 2 # This program repeats Selection12.py from Chapter 7.
 3 # It is another example of "Nested Control Structures",
 4 # in this case, "Nested Selection".
 5
 6
 7 print()
 8 sat = eval(input("Enter SAT score  -->  "))
 9 print()
10
11 if sat >= 1100:
12     print("You are admitted.")
13     print("Orientation will start in June.")
14     print()
15     income = eval(input("Enter your family income  -->  "))
16     print()
17     if income < 20000:
18         print("You qualify for financial aid.")
19     else:
20         print("You do not qualify for financial aid.")
21 else:
22     print("You are not admitted.")
23     print("Please try again when your SAT improves.")
```


----jGRASP exec: python Selection12.py
Enter SAT score --> 1350
You are admitted.
Orientation will start in June.
Enter your family income --> 18000
You qualify for financial aid.
----jGRASP: operation comp
----jGRASP exec: python Se
Enter SAT score --> 1500
You are admitted.
Orientation will start in June.
Enter your family income --> 90000
You do not qualify for financial aid.
----jGRASP: operation complete.
----jGRASP exec: python Selection12.py
Enter SAT score --> 700
You are not admitted.
Please try again when your SAT improves.
----jGRASP: operation complete.

```
 1 # Nested06.py
 2 # This program demonstrates that control structures
 3 # can be nested with more than 2 levels.
 4 # This program is actually the entire previous
 5 # program nested inside a loop that repeats 5 times.
 6 # NOTE: As you have more and more levels of nesting
 7 #       indentation becomes more and more important.
 8 # ALSO: This program does have an issue in that it
 9 #       basically assumes you will always interview
10 #       exactly 5 students.
11
12
13 for k in range(5):
14     print()
15     sat = eval(input("Enter SAT score  -->  "))
16     print()
17
18     if sat >= 1100:
19         print("You are admitted.")
20         print("Orientation will start in June.")
21         print()
22         income = eval(input("Enter your family income  -->  "))
23         print()
24         if income < 20000:
25             print("You qualify for financial aid.")
26         else:
27             print("You do not qualify for financial aid.")
28     else:
29         print("You are not admitted.")
30         print("Please try again when your SAT improves.")
31
32     print("\n-------------------------------------")
```

```python
# Nested06.py
# This program demonstrates that control struc
# can be nested with more than 2 levels.
# This program is actually the entire previous
# program nested inside a loop that repeats 5
# NOTE: As you have more and more levels of ne
#       indentation becomes more and more impo
# ALSO: This program does have an issue in tha
#       basically assumes you will always inte
#       exactly 5 students.


for k in range(5):
    print()
    sat = eval(input("Enter SAT score  -->  "))
    print()

    if sat >= 1100:
        print("You are admitted.")
        print("Orientation will start in June.")
        print()
        income = eval(input("Enter your family i
        print()
        if income < 20000:
            print("You qualify for financial aid.
        else:
            print("You do not qualify for financi
    else:
        print("You are not admitted.")
        print("Please try again when your SAT im

    print("\n-----------------------------
```

```
Enter SAT score  -->  900

You are not admitted.
Please try again when your SAT improves.

-----------------------------------

Enter SAT score  -->  1000

You are not admitted.
Please try again when your SAT improves.

-----------------------------------

Enter SAT score  -->  1099

You are not admitted.
Please try again when your SAT improves.

-----------------------------------

Enter SAT score  -->  1100

You are admitted.
Orientation will start in June.

Enter your family income  -->  18000

You qualify for financial aid.

-----------------------------------

Enter SAT score  -->  1200

You are admitted.
Orientation will start in June.

Enter your family income  -->  150000

You do not qualify for financial aid.

-----------------------------------
```

```
# Nested07.py
# This program is very similar to the previous
# program.  The different is that it begins with
# an input statement that allows the interviewer
# to enter the number of students that he/she
# needs to interview.


print()
numStudents = eval(input("How many students do you need to interview?  -->  "))

for k in range(numStudents):
   print()
   sat = eval(input("Enter SAT score  -->  "))
   print()

   if sat >= 1100:
      print("You are admitted.")
      print("Orientation will start in June.")
      print()
      income = eval(input("Enter your family income  -->  "))
      print()
      if income < 20000:
         print("You qualify for financial aid.")
      else:
         print("You do not qualify for financial aid.")
   else:
      print("You are not admitted.")
      print("Please try again when your SAT improves.")

   print("\n-----------------------------------")
```

```python
# Nested07.py
# This program is very similar to the pr
# program.  The different is that it beg
# an input statement that allows the int
# to enter the number of students that h
# needs to interview.


print()
numStudents = eval(input("How many

for k in range(numStudents):
    print()
    sat = eval(input("Enter SAT score  --
    print()

    if sat >= 1100:
        print("You are admitted.")
        print("Orientation will start in J
        print()
        income = eval(input("Enter your fa
        print()
        if income < 20000:
            print("You qualify for financia
        else:
            print("You do not qualify for f
    else:
        print("You are not admitted.")
        print("Please try again when your
    print("\n----------------------------
```

```
How many students do you need to interview?  -->  3

Enter SAT score  -->  1300

You are admitted.
Orientation will start in June.

Enter your family income  -->  19000

You qualify for financial aid.

-----------------------------------

Enter SAT score  -->  1500

You are admitted.
Orientation will start in June.

Enter your family income  -->  99000

You do not qualify for financial aid.

-----------------------------------

Enter SAT score  -->  700

You are not admitted.
Please try again when your SAT improves.

-----------------------------------
```

```
# Nested08.py
# This program fixes the issue of the previous program.
# Now everything is inside a <while> loop.
# At the conclusion of each interview the user has
# the option to repeat the program.
# The <while> loop makes the program repeat as long
# as the user responds with a capital 'Y'.


response = 'Y';

while response == 'Y':  # Note: Only capital 'Y' will make the loop repeat.
   print()
   sat = eval(input("Enter SAT score  -->  "))
   print()

   if sat >= 1100:
      print("You are admitted.")
      print("Orientation will start in June.")
      print()
      income = eval(input("Enter your family income  -->  "))
      print()
      if income < 20000:
         print("You qualify for financial aid.")
      else:
         print("You do not qualify for financial aid.")
   else:
      print("You are not admitted.")
      print("Please try again when your SAT improves.")

   print()
   response = input("Do you want to interview another student?  {Y/N}  -->  ")
```

```
1  # Nested08.py
2  # This program fixes the issue
3  # Now everything is inside a <wh
4  # At the conclusion of each int
5  # the option to repeat the progr
6  # The <while> loop makes the pro
7  # as the user responds with a ca
8
9
10 response = 'Y';
11
12 while response == 'Y':   #
13    print()
14    sat = eval(input("Enter SAT s
15    print()
16
17    if sat >= 1100:
18       print("You are admitted."
19       print("Orientation will st
20       print()
21       income = eval(input("Enter
22       print()
23       if income < 20000:
24          print("You qualify for
25       else:
26          print("You do not quali
27    else:
28       print("You are not admitte
29       print("Please try again wh
30
31    print()
32    response = input("Do you
```

```
Enter SAT score  -->  1300

You are admitted.
Orientation will start in June.

Enter your family income  -->  19000

You qualify for financial aid.

Do you want to interview another student?  {Y/N}  -->  Y

Enter SAT score  -->  1500

You are admitted.
Orientation will start in June.

Enter your family income  -->  99000

You do not qualify for financial aid.

Do you want to interview another student?  {Y/N}  -->  Y

Enter SAT score  -->  700

You are not admitted.
Please try again when your SAT improves.

Do you want to interview another student?  {Y/N}  -->  N
```

```
1  # Nested09.py
2  # This program demonstrates that repetition can
3  # be nested inside a selection.
4  # In truth, ANY control structure can be nested
5  # inside ANY other control structure.
6  # The program also shows how to determine if a
7  # number is even or odd.
8
9
10 print()
11 stop = eval(input("Enter a number between 1 and 15.  -->  "))
12 print()
13
14 if (stop % 2 == 0):  # if stop is even
15     for k in range(stop):
16         print("EVEN",end = " ")
17 else:                # if stop is odd
18     for k in range(stop):
19         print("ODD",end = " ")
20
21 print()
```

```
  ----jGRASP exec: python Nested09.py

Enter a number between 1 and 15.  -->  10

EVEN EVEN EVEN EVEN EVEN EVEN EVEN EVEN EVEN EVEN

  ----jGRASP: operation complete.
```

```
10 print()
11 stop = eval(input("Enter a number between 1 and 15.  -->  "))
12 print()
13
14 if (stop % 2 == 0):  # if stop is even
15     for k in range(stop):
16         print("EVEN",end = " ")
17 else:                 # if stop is odd
18     for k in range(stop):
19         print("ODD",end = " ")
20
21 print()
```

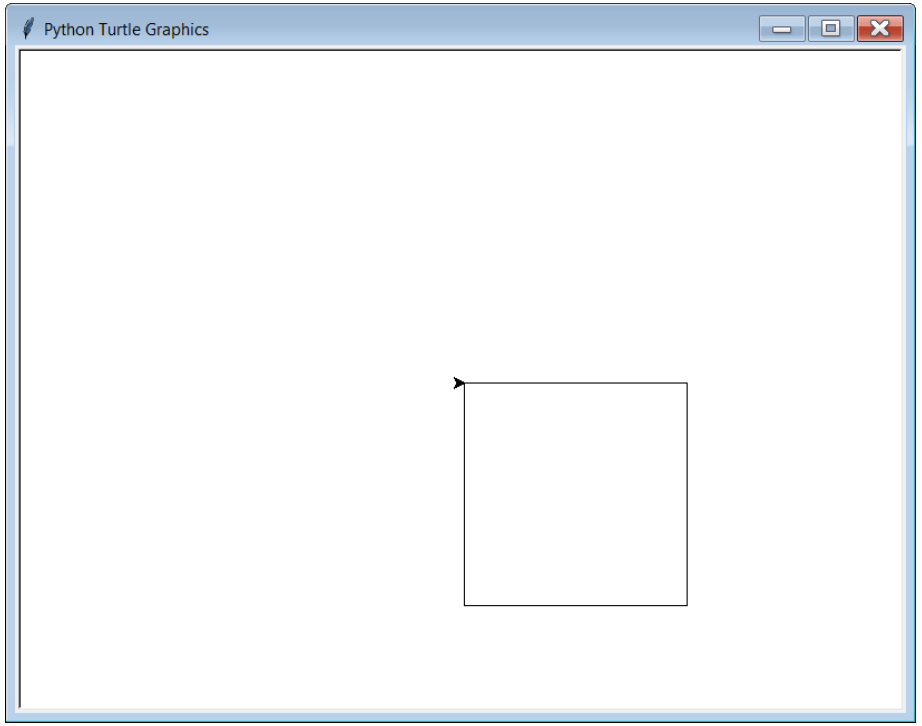
```
----jGRASP exec: python Nested09.py

Enter a number between 1 and 15.  -->  13

ODD ODD ODD ODD ODD ODD ODD ODD ODD ODD ODD ODD ODD

----jGRASP: operation complete.
```

```
10 print()
11 stop = eval(input("Enter a number between 1 and 15.  -->  "))
12 print()
13
14 if (stop % 2 == 0):  # if stop is even
15     for k in range(stop):
16         print("EVEN",end = " ")
17 else:                 # if stop is odd
18     for k in range(stop):
19         print("ODD",end = " ")
20
21 print()
```

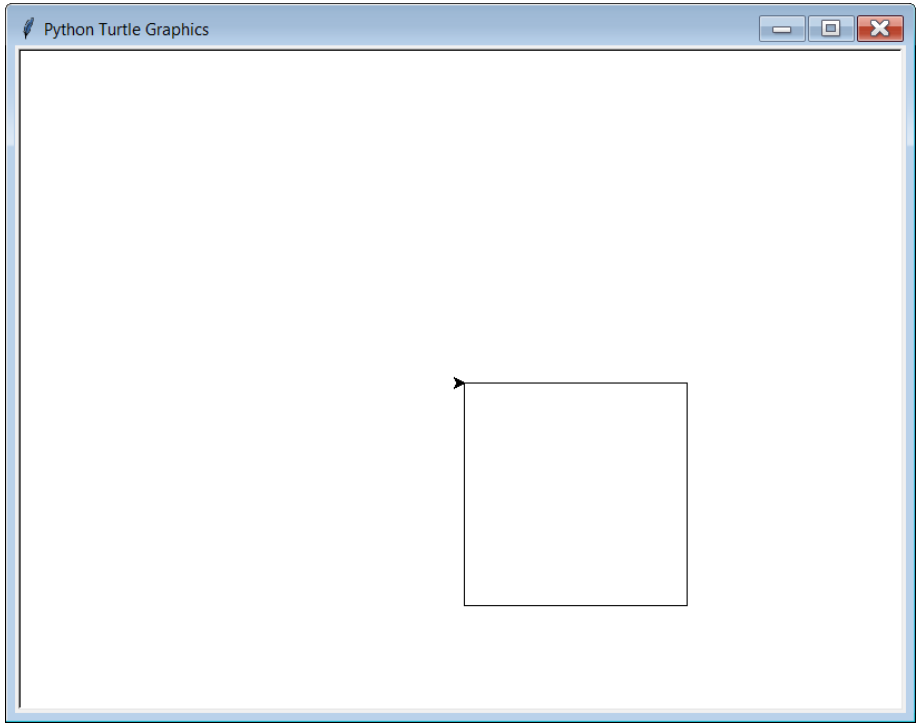
Section 8.5

Using Repetition with Turtle Graphics

```
 1 # RepetitionWithGraphics01.py
 2 # This program uses repeats TurtleGraphics08.py
 3 # to demonstrate an inefficient way to draw
 4 # a square.
 5
 6
 7 from turtle import *
 8
 9 setup(800,600)
10
11 forward(200)
12 right(90)
13 forward(200)
14 right(90)
15 forward(200)
16 right(90)
17 forward(200)
18 right(90)
19
20 update()
21 done()
```

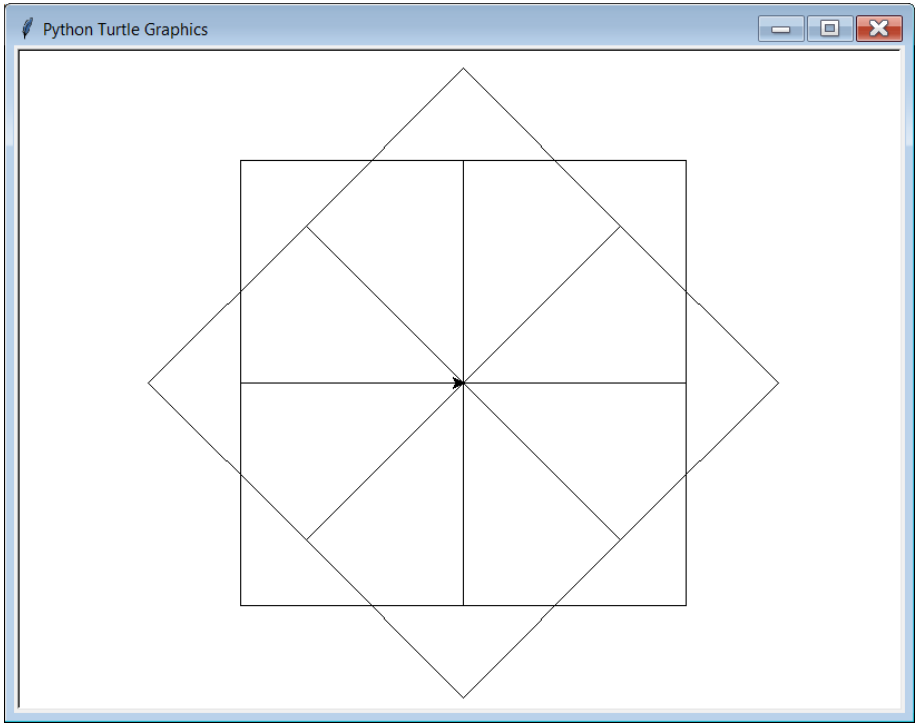


```
1  # RepetitionWithGraphics02.py
2  # This program draws the same square as the previous
3  # program, but is more efficient because it uses a
4  # <for> loop to create the square.
5
6
7  from turtle import *
8
9  setup(800,600)
10
11 for k in range(4):
12     forward(200)
13     right(90)
14
15 update()
16 done()
```

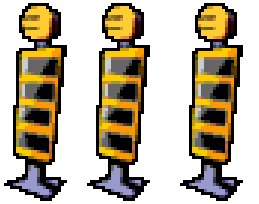


```
1  # RepetitionWithGraphics03.py
2  # This program takes the "square loop" from the
3  # previous program and "nests" it inside another
4  # <for> loop to create a special design.
5
6
7  from turtle import *
8
9  setup(800,600)
10
11 for j in range(8):
12     for k in range(4):
13         forward(200)
14         right(90)
15     left(45)
16
17 update()
18 done()
```

```
1  # RepetitionWithGraphics03.py
2  # This program takes the "square loop" from the
3  # previous program and "nests" it inside another
4  # <for> loop to create a special design.
5
6
7  from turtle import *
8
9  setup(800,600)
10
11 for j in range(8):
12     for k in range(4):
13         forward(200)
14         right(90)
15     left(45)
16
17 update()
18 done()
```



Lab 8A



What you saw in the last couple program examples relates directly to what you will be doing in Lab 8A.