Exposure CS 2021 for CS1

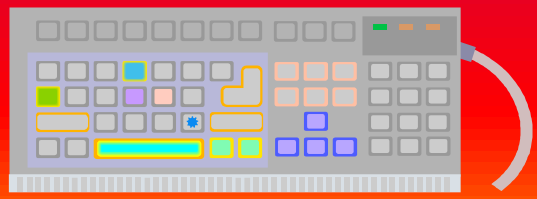
Chapter 7 Section 1-5 Slides

Keyboard Input and 1-Way Selection Control Structures

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

Section 7.1

Keyboard Input



```
 1 # KeyboardInput01.py
 2 # This program demonstrates a very user-UNfriendly
 3 # way to do keyboard input.
 4 # The user has no idea what he/she is entering.
 5
 6
 7 print()
 8
 9 name = input()
10
11 print("Hello",name)
12
```

```
----jGRASP exec: python KeyboardInput01.py
```

Initial Output: All we see is a blinking cursor.

```
 1 # KeyboardInput01.py
 2 # This program demonstrates a very user-UNfriendly
 3 # way to do keyboard input.
 4 # The user has no idea what he/she is entering.
 5
 6
 7 print()
 8
 9 name = input()
10
11 print("Hello",name)
12
```

```
    ----jGRASP exec:

John
Hello John

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

If the user has knowledge of the inner workings of the program (something that cannot be assumed) he/she will enter his/her name and then the program execution can continue.

To prove that the name was entered, it is displayed in the form of a greeting.

```
 1 # KeyboardInput02.py
 2 # This program improves the keyboard input
 3 # by adding a user-friendly "prompt".
 4
 5
 6 print()
 7
 8 name = input("Please enter your name.  -->  ")
 9
10 print("Hello",name)
```

```
    ----jGRASP exec: python KeyboardInput02.py

▶▶  Please enter your name.  -->
```

When you first run the program, all you see is the *prompt*.

```
1  # KeyboardInput02.py
2  # This program improves the keyboard input
3  # by adding a user-friendly "prompt".
4
5
6  print()
7
8  name = input("Please enter your name.  -->  ")
9
10 print("Hello",name)
```

```
    ----jGRASP exec: python KeyboardInput02.py

▶▶  Please enter your name.  -->  John
```

Then you type your name.

```
1  # KeyboardInput02.py
2  # This program improves the keyboard input
3  # by adding a user-friendly "prompt".
4
5
6  print()
7
8  name = input("Please enter your name.  -->  ")
9
10 print("Hello",name)
```

```
  ----jGRASP exec: python KeyboardInput02.py

▶▶ Please enter your name.  -->  John
   Hello John

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

When you press <enter> you see the rest of the program output.

```
1  # KeyboardInput02.py
2  # This program improves the keyboard input
3  # by adding a user-friendly "prompt".
4
5
6  print()
7
8  name = input("Please enter your name.  -->  ")
9
10 print("Hello",name)
```

```
   ----jGRASP exec: python KeyboardInput02.py

▶▶ Please enter your name.  -->  Diana
   Hello Diana

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

Enter a different name – get a different output.

```
1  # KeyboardInput03.py
2  # This program enters 3 different names: <firstName>,
3  # <middleName> and <lastName>; on 3 different lines
4  # and then combines them all into a full name.
5  # NOTE: You will not see the second prompt until you
6  # finish entering the information from the first and
7  # press <enter>.
8
9
10 print()
11
12 firstName  = input("Please enter your first name.   -->  ")
13 middleName = input("Please enter your middle name.  -->  ")
14 lastName   = input("Please enter your last name.    -->  ")
15
16 fullName = firstName + " " + middleName + " " + lastName
17
18 print()
19 print("Your full name is",fullName)
20
```

```
----jGRASP exec: python KeyboardInput03.py

Please enter your first name.    -->  John
Please enter your middle name.   -->  Quincy
Please enter your last name.     -->  Public

Your full name is John Quincy Public

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

```
12 firstName  = input("Please enter your first name.   -->  ")
13 middleName = input("Please enter your middle name.  -->  ")
14 lastName   = input("Please enter your last name.    -->  ")
15
16 fullName = firstName + " " + middleName + " " + lastName
17
18 print()
19 print("Your full name is",fullName)
20
```

```
 1 # KeyboardInput04.py
 2 # This program attempts to compute  the sum of two
 3 # numbers entered by the user.  The problem is the
 4 # numbers are being entered as strings instead of
 5 # numbers.
 6
 7
 8 print()
 9
10 num1 = input("Please enter the 1st number.  -->  ")
11 num2 = input("Please enter the 2nd number.  -->  ")
12
13 sum = num1 + num2
14
15 print()
16 print("The sum of",num1,"and",num2,"is",sum)
17
```

```
----jGRASP exec: python KeyboardInput04.py

Please enter the 1st number.  -->  100
Please enter the 2nd number.  -->  200

The sum of 100 and 200 is 100200

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

```
10 num1 = input("Please enter the 1st number.  -->  ")
11 num2 = input("Please enter the 2nd number.  -->  ")
12
13 sum = num1 + num2
14
15 print()
16 print("The sum of",num1,"and",num2,"is",sum)
17
```

Addition vs. Concatenation

The problem with the previous program is that, by itself, the input command returns a *string* value.

When the plus sign (+) is used with these string values, we get *String Concatenation* instead of addition.

If you want keyboard input to be evaluated as a number, you need to use the eval command with input. This will give you number input.

```
1  # KeyboardInput05.py
2  # This program demonstrates the proper way to enter
3  # numerical input by using the <eval> function.
4  # The function makes the computer EVALuate the
5  # input to see what kind of information it is
6  # and properly identify numerical input.
7
8
9  print()
10
11 num1 = eval(input("Please enter the 1st number.  -->  "))
12 num2 = eval(input("Please enter the 2nd number.  -->  "))
13
14 sum = num1 + num2
15
16 print()
17 print("The sum of",num1,"and",num2,"is",sum)
18
```

```
----jGRASP exec: python KeyboardInput05.py

Please enter the 1st number.  -->  100
Please enter the 2nd number.  -->  200

The sum of 100 and 200 is 300

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

```
11 num1 = eval(input("Please enter the 1st number.  -->  "))
12 num2 = eval(input("Please enter the 2nd number.  -->  "))
13
14 sum = num1 + num2
15
16 print()
17 print("The sum of",num1,"and",num2,"is",sum)
18
```

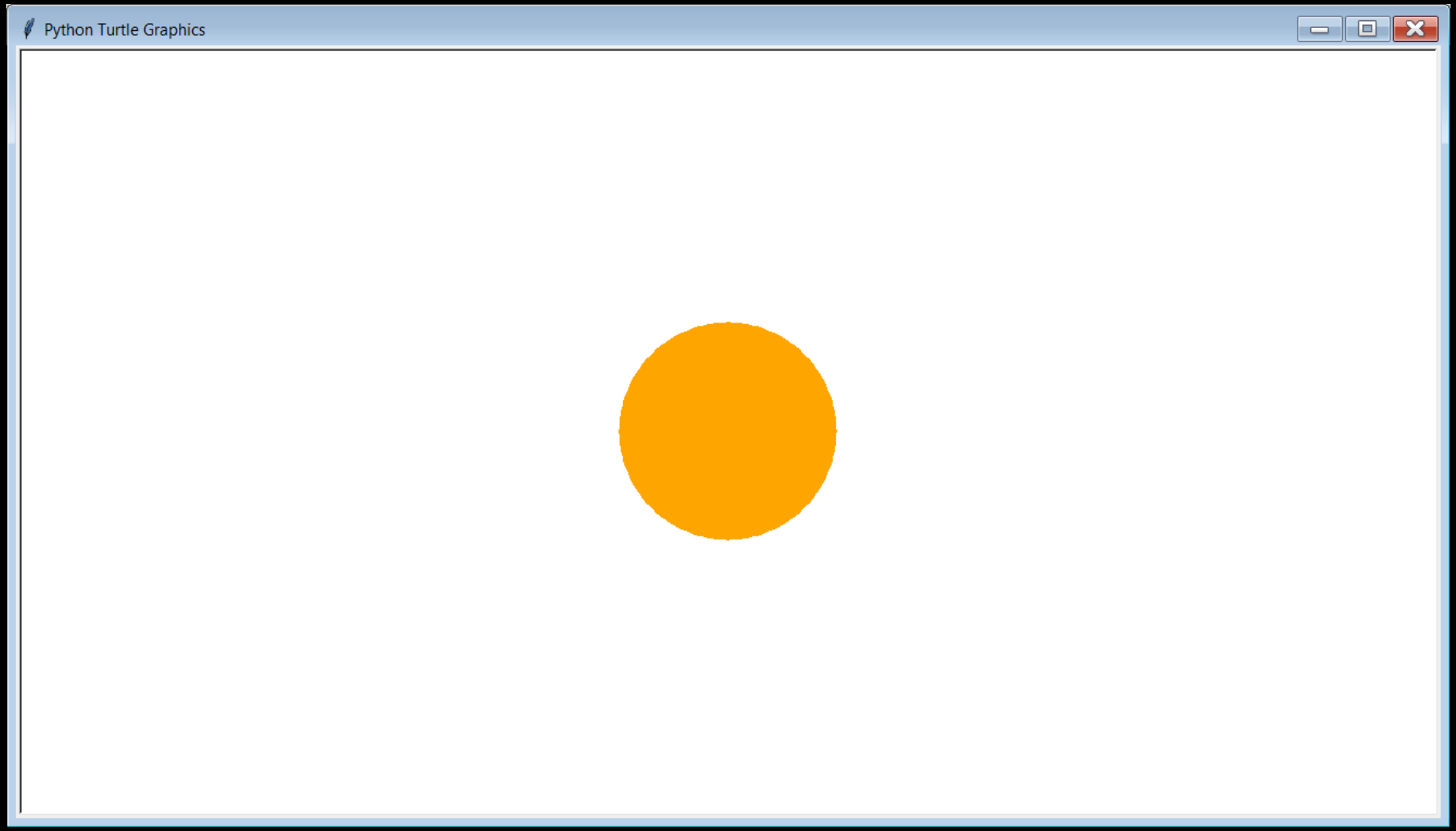
```
 1 # KeyboardInput06.py
 2 # This program computes the average of 3 numbers
 3 # entered by the user.  Note that this works for
 4 # both integers and real numbers.
 5
 6
 7 print()
 8
 9 num1 = eval(input("Please enter the 1st number.  -->  "))
10 num2 = eval(input("Please enter the 2nd number.  -->  "))
11 num3 = eval(input("Please enter the 3rd number.  -->  "))
12
13 average = (num1 + num2 + num3) / 3
14
15 print()
16 print("The average of",num1,"and",num2,"and",num3,"is",
average)
17
```

```
   ----jGRASP exec: python KeyboardInput06.py

▶▶ Please enter the 1st number.   -->  22.22
▶▶ Please enter the 2nd number.   -->  33.33
▶▶ Please enter the 3rd number.   -->  77.77

   The average of 22.22 and 33.33 and 77.77 is 44.44

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

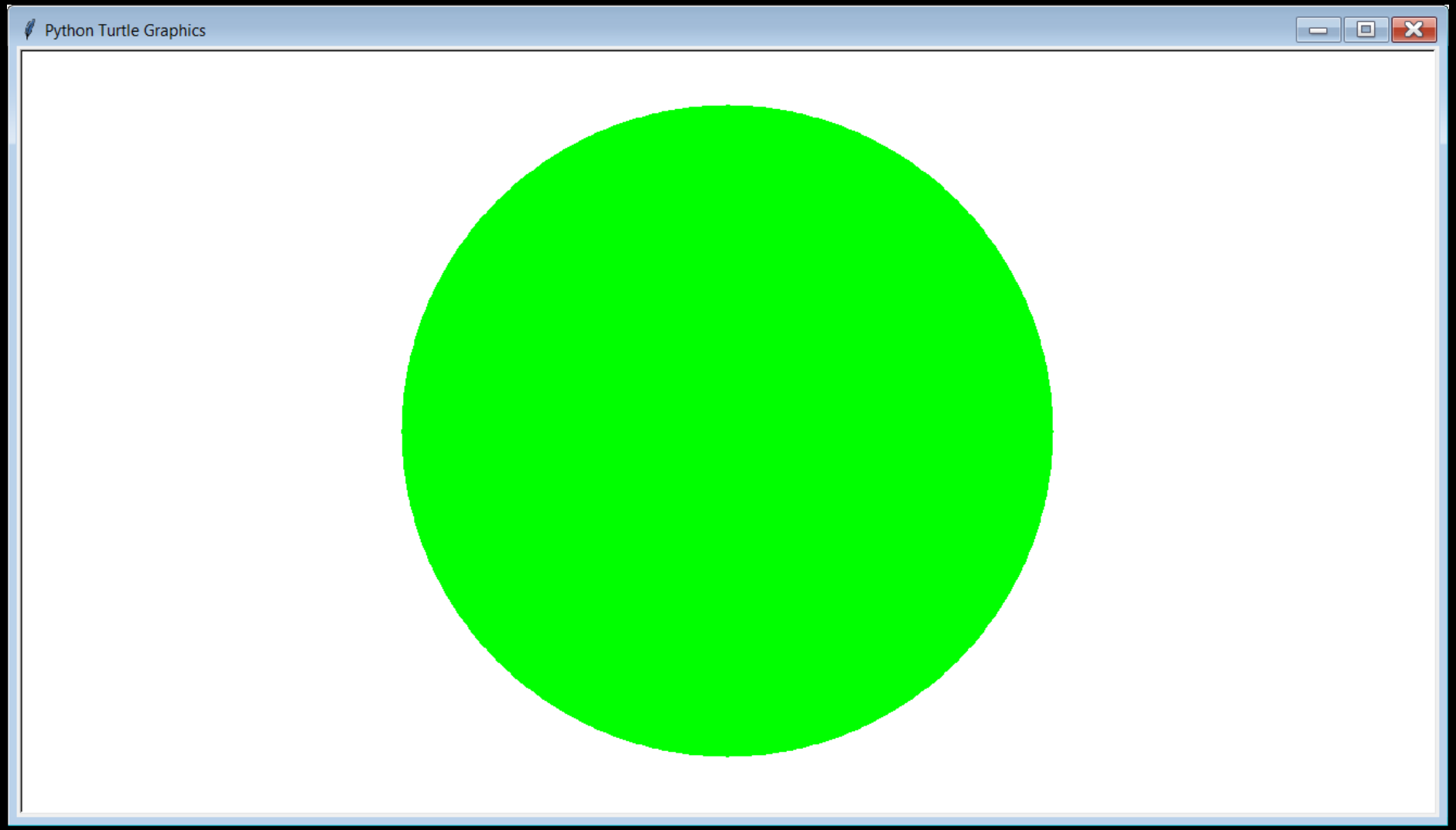
```
 9 num1 = eval(input("Please enter the 1st number.   -->  "))
10 num2 = eval(input("Please enter the 2nd number.   -->  "))
11 num3 = eval(input("Please enter the 3rd number.   -->  "))
12
13 average = (num1 + num2 + num3) / 3
14
15 print()
16 print("The average of",num1,"and",num2,"and",num3,"is",
average)
17
```

```
1 # KeyboardInput07.py
2 # This program demonstrates <input> can be used with
3 # graphics.  While it technically works, it is awkward
4 # as you fumble between the text input in the editor
5 # and the graphics output in the graphics window.
6
7
8 from Graphics import *
9
10 beginGrfx(1300,700)
11
12 myColor = input("Enter any color name.  -->  ")
13 myRadius = eval(input("Enter a radius value from 1-300.  -->  "))
14 setColor(myColor)
15 fillCircle(650,350,myRadius)
16
17 endGrfx()
18
```

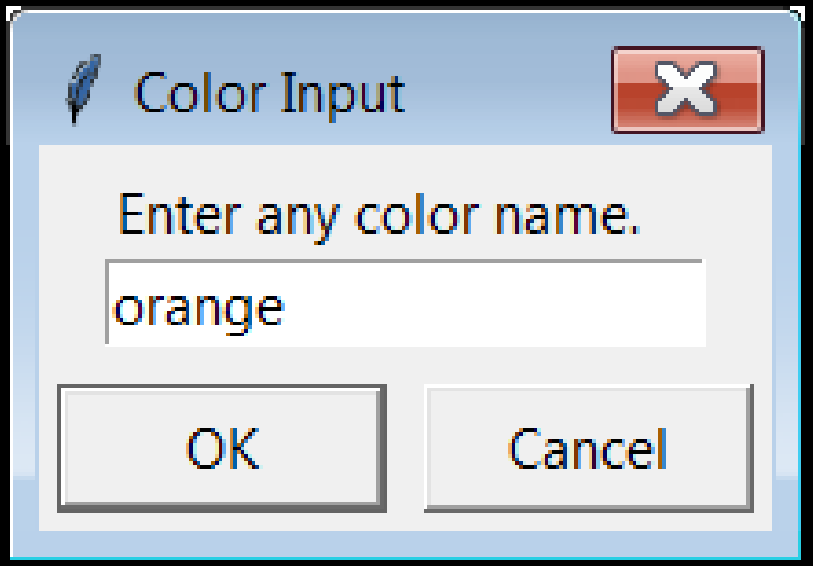
```
----jGRASP exec: python KeyboardInput07.py
Enter any color name.  -->  orange
Enter a radius value from 1-300.  -->  100
```

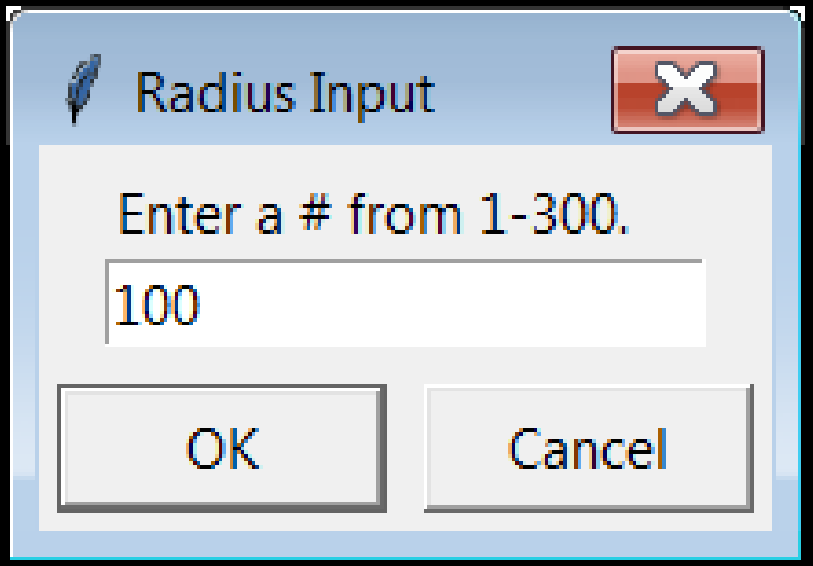


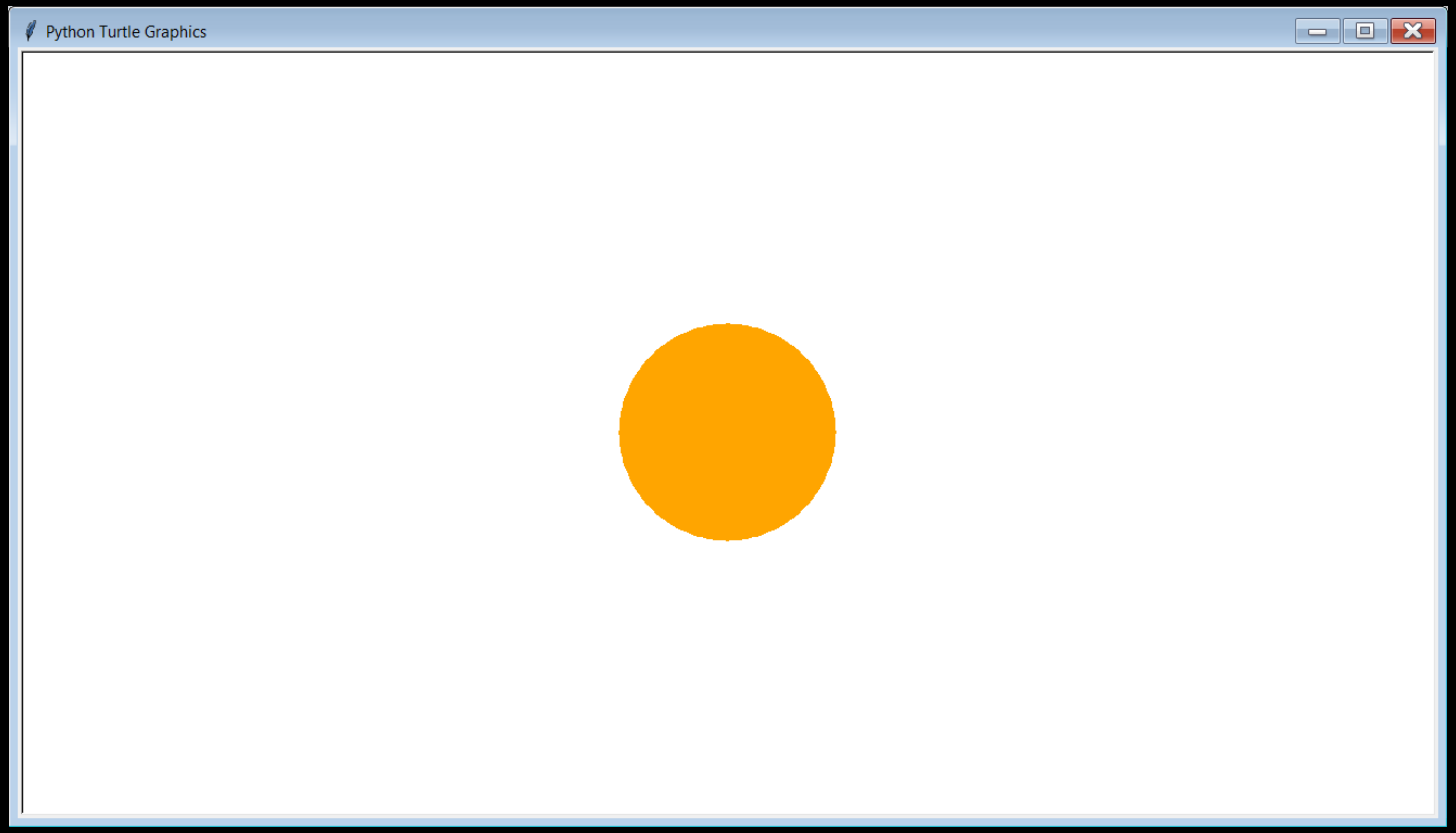
```
----jGRASP exec: python KeyboardInput07.py
Enter any color name.  -->  green
Enter a radius value from 1-300.  -->  300
```

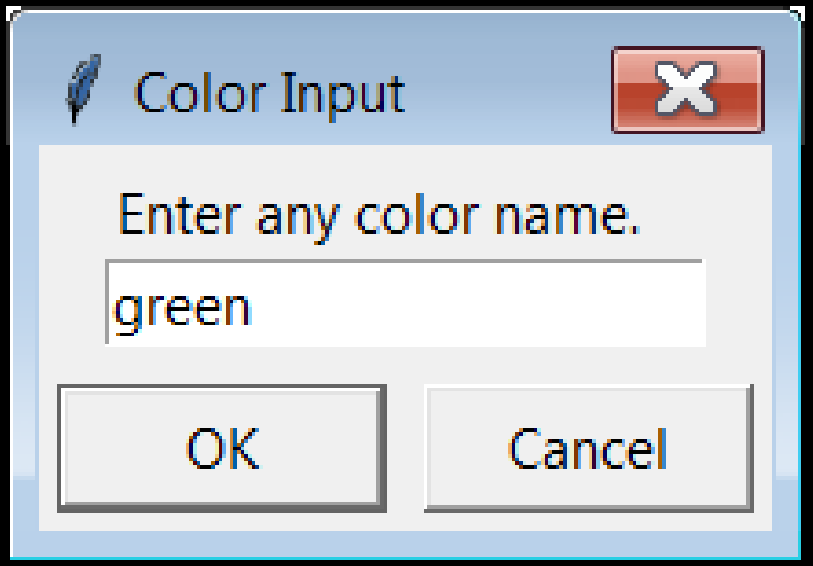


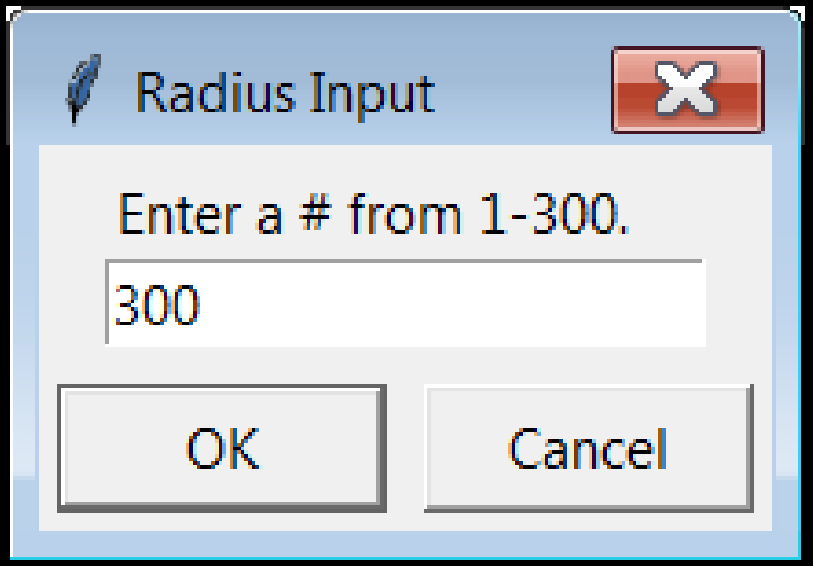
```
1 # KeyboardInput08.py
2 # This program demonstrates <textinput> and <numinput>
3 # which are better suited for graphics programs.
4 # Note that <numinput> does not require the <eval> function.
5
6
7 from Graphics import *
8
9 beginGrfx(1300,700)
10
11 myColor = textinput("Color Input","Enter any color name.")
12 myRadius = numinput("Radius Input","Enter a # from 1-300.")
13 setColor(myColor)
14 fillCircle(650,350,myRadius)
15
16 endGrfx()
17
```

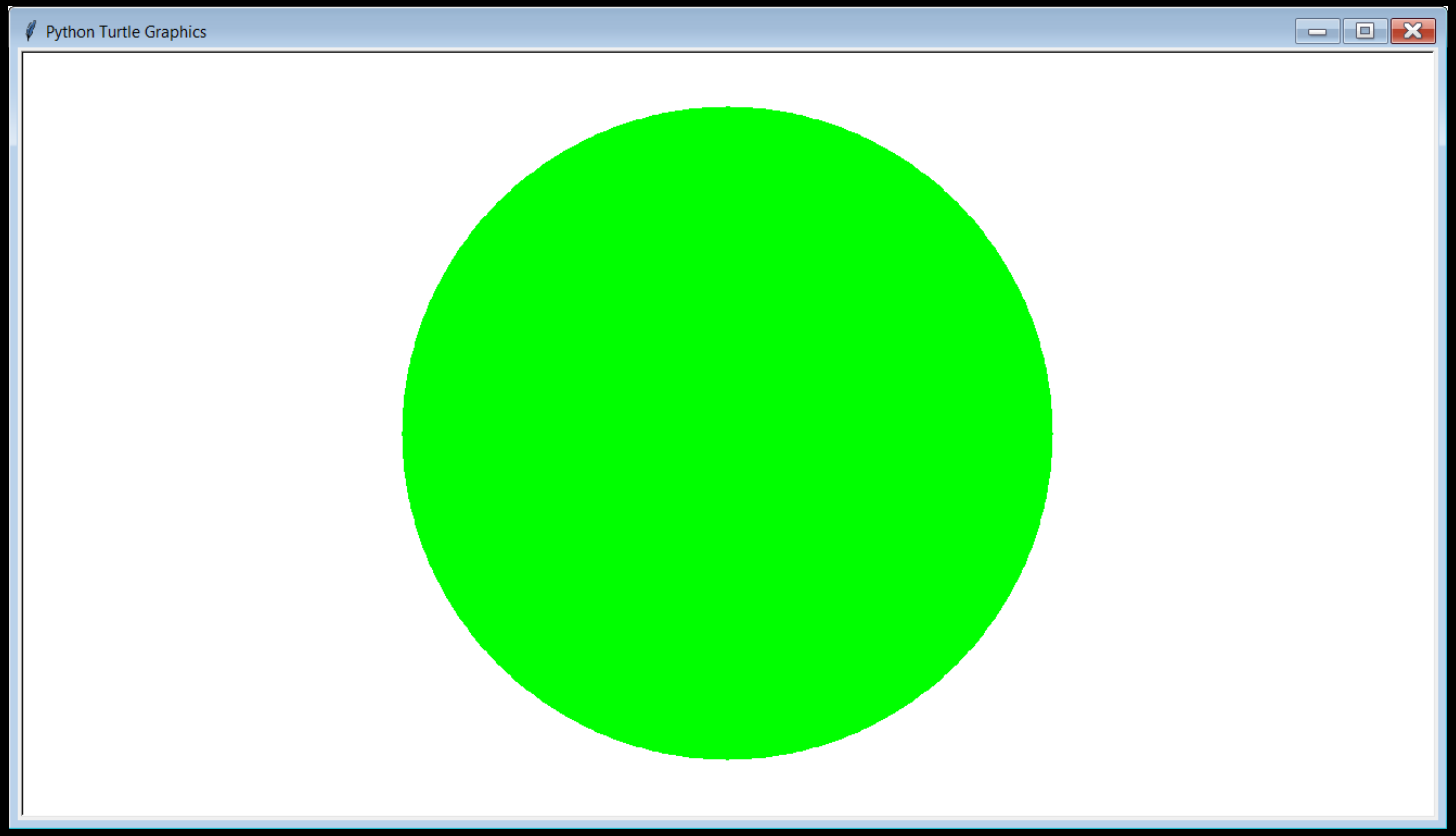

Color Input
Enter any color name.
orange
OK
Cancel


Radius Input
Enter a # from 1-300.
100
OK
Cancel


Python Turtle Graphics


Color Input
Enter any color name.
green
OK
Cancel


Radius Input
Enter a # from 1-300.
300
OK
Cancel


Python Turtle Graphics

Input Functions

input(*prompt***)**
is used for string input on the text screen.

eval(input(*prompt***))**
is used for number input on the text screen.

textinput(*title,prompt***)**
is used for string input on the graphics window.

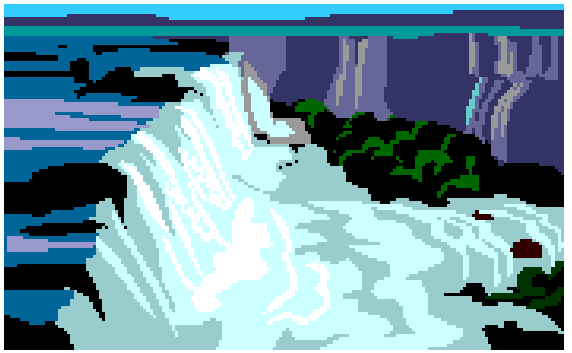
numinput(*title,prompt***)**
is used for number input on the graphics window.

Section 7.2

Introduction to Control Structures

Program Flow

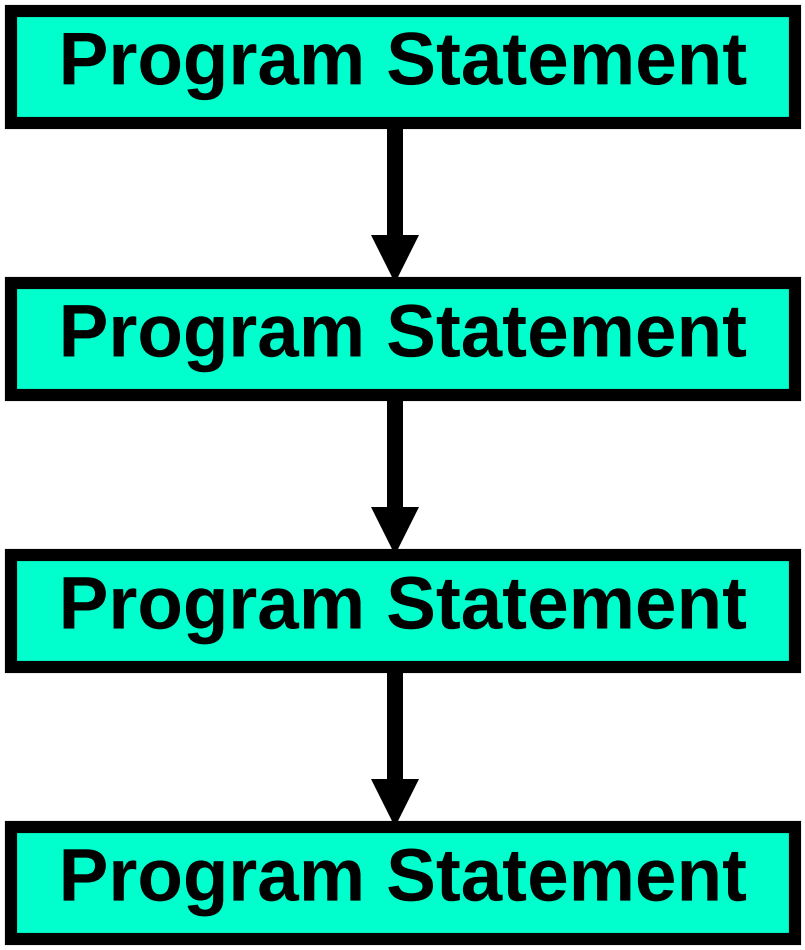
Program Flow **follows the exact sequence of listed program statements, unless directed otherwise by a Python control structure.**



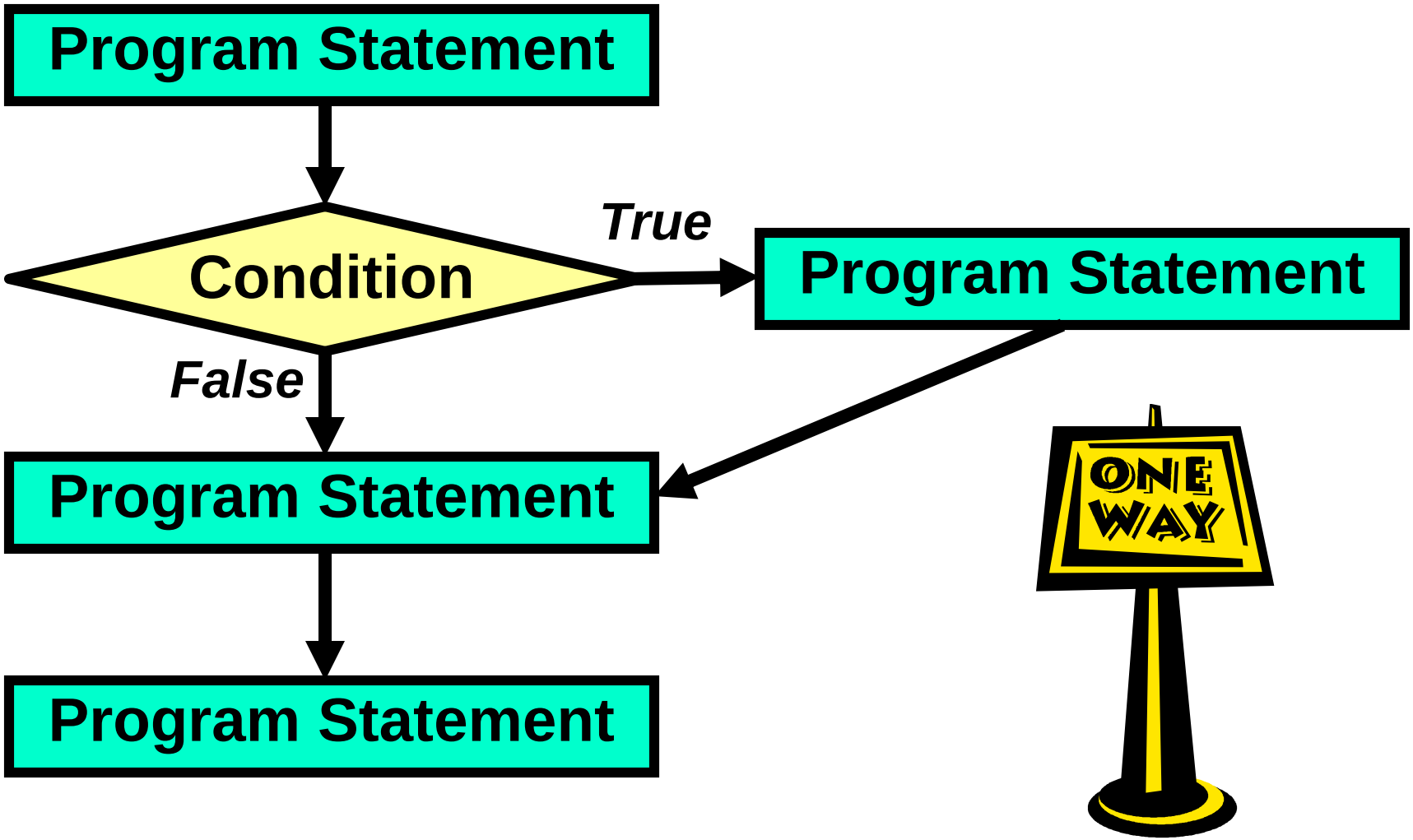
Section 7.3

Types of Control Structures

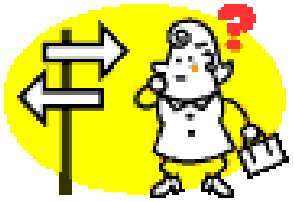
Simple Sequence



One-Way Selection

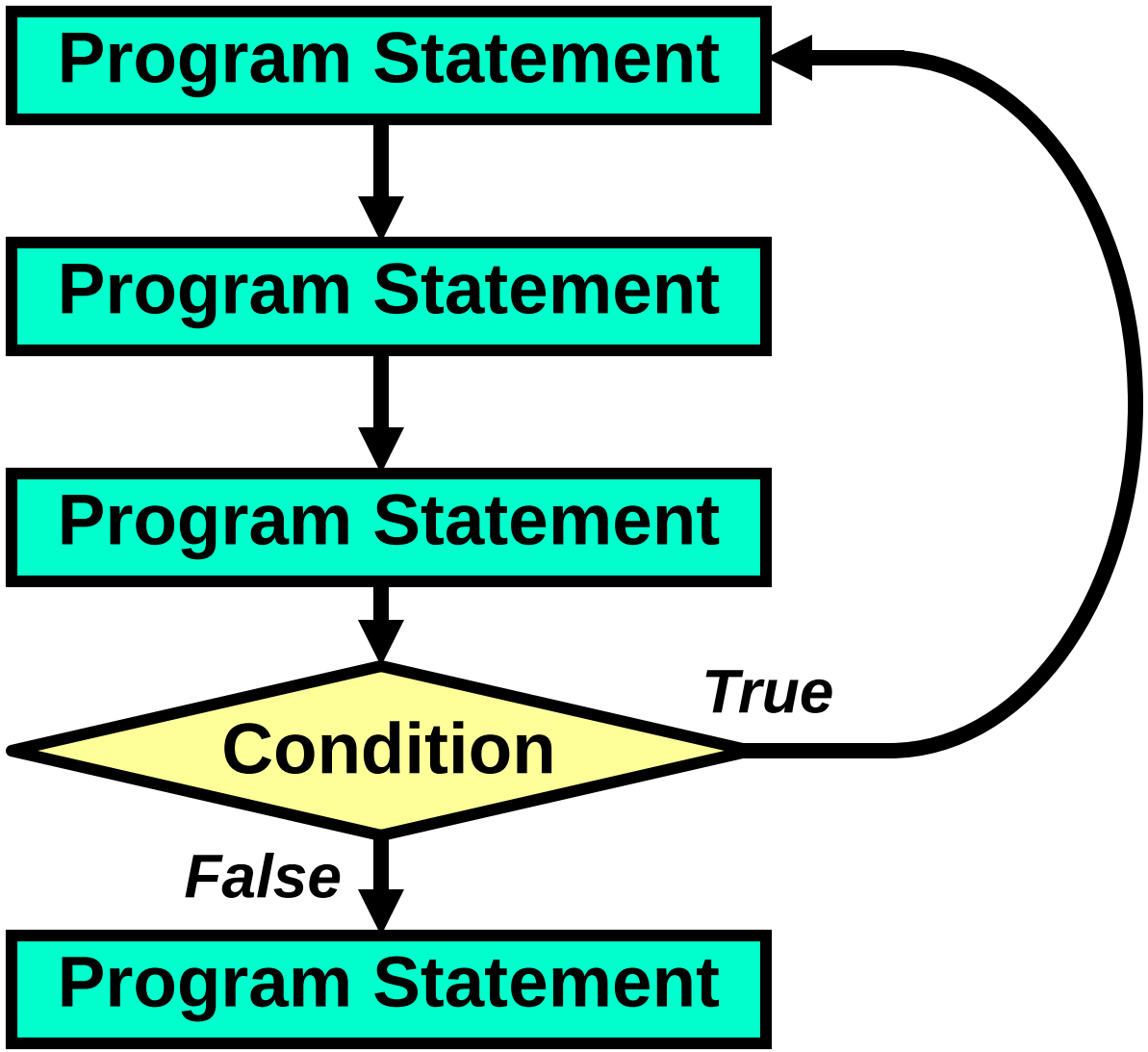


Two-Way Selection



Multiple-Way Selection

Repetition


Program Statement
Program Statement
Program Statement
Condition
True
False
Program Statement

Types of Control Structures

- **Simple Sequence**
- **Selection** ***also called:***
  - **Decision Making**
  - **Conditional Branching**
  - **Alternation**
- **Repetition** ***also called:***
  - **Looping**
  - **Iteration**

Conditional Statement Definition

A *conditional statement* is a program expression that evaluates to True or False.

Most conditional statements require a *relational operator*.

All conditions end with a colon (:).

Section 7.4

Relational Operators

Relational Operators

Name	Operator	Expression	Evaluates
Equals	==	5 == 5 5 == 10	True False
Not Equals	!=	50 != 25 100 != 100	True False
Less than	<	100 < 200 200 < 100	True False
Greater than	>	200 > 100 200 > 200	True False
Less than or equals	<=	100 <= 200 200 <= 200 200 <= 100	True True False
Greater than or equals	>=	100 >= 200 200 >= 200 200 >= 100	False True True

Important Note:

The relational operators shown on the previous slide will be used in the Python example programs that demonstrate the different control structures.

Be careful not to confuse the equality operator (==) with the assignment operator (=).

Assignment (=)

```
x = 10
```

Assigns a the value of **10** to **x**.

Equality (==)

```
if x == 10:
```

Checks if **x** is equal to **10**.

Section 7.5

Selection

```
 1 # Selection01.py
 2 # This program demonstrates one-way selection
 3 # with <if>.  Run the program twice.
 4 # First with <sales> equal to 300,000 and a
 5 # second time with <sales> equal to 500,000.
 6
 7
 8 print()
 9 sales = eval(input("Enter Sales  -->  "))
10 bonus = 0
11
12 if sales >= 500000:
13     bonus = 1000
14
15 print()
16 print("Christmas bonus:",bonus)
```

```
  ----jGRASP exec: python

 Enter Sales  -->  300000

 Christmas bonus: 0

  ----jGRASP: operation co
```

trates one-way selection
program twice.
equal to 300,000 and a
ales> equal to 500,000.

```
 8 print()
 9 sales = eval(input("Enter Sales  -->  "))
10 bonus = 0
11
12 if sales >= 500000:
13     bonus = 1000
14
15 print()
16 print("Christmas bonus:",bonus)
```

```
  ----jGRASP exec: python

 Enter Sales  -->  300000

 Christmas bonus: 0

  ----jGRASP: operation co
```

```
  ----jGRASP exec: python

 Enter Sales  -->  500000

 Christmas bonus: 1000

  ----jGRASP: operation co
```

```
 8 print()
 9 sales = eval(input("Enter Sales  -->  "))
10 bonus = 0
11
12 if sales >= 500000:
13     bonus = 1000
14
15 print()
16 print("Christmas bonus:",bonus)
```

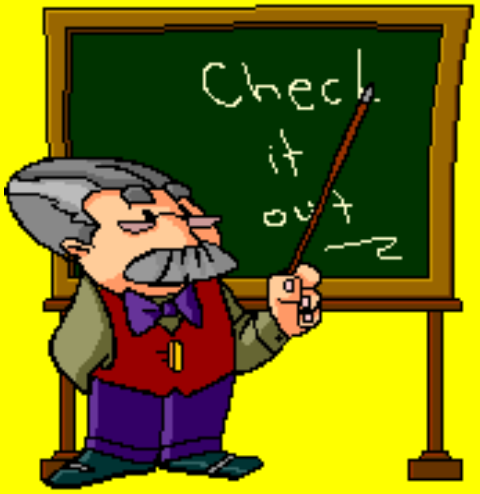
```
1  # Selection02.py
2  # This program demonstrates the Syntax Error
3  # you receive when you do not properly indent
4  # the programming statement(s) being controlled
5  # by a control structure.
6
7  # NOTE: In most languages, indentation is recommended.
8  #       In Python, indentation is required.
9
10
11 print()
12 sales = eval(input("Enter Sales  -->  "))
13 bonus = 0
14
15 if sales >= 500000:
16 bonus = 1000
17
18 print()
19 print("Christmas bonus:",bonus)
```

```
----jGRASP exec: python Selection02.py
  File "Selection02.py", line 16
    bonus = 1000
          ^
IndentationError: expected an indented block

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

```
11 print()
12 sales = eval(input("Enter Sales  -->  "))
13 bonus = 0
14
15 if sales >= 500000:
16 bonus = 1000
17
18 print()
19 print("Christmas bonus:",bonus)
```

Indentation Rule:

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 # Selection03.py
2 # This program demonstrates a control structure
3 # can control multiple programming commands as
4 # long as proper, consistent indentation is used.
5
6
7 print()
8 sales = eval(input("Enter Sales  -->  "))
9 bonus = 0
10
11 if sales >= 500000:
12    print("\nCONGRATULATIONS!")
13    print("You sold half a million dollars in merchandise!")
14    print("You will receive a $1000 Christmas Bonus!")
15    print("Keep up the good work!")
16    bonus = 1000
17
18 print()
19 print("Christmas bonus:",bonus)
20
```

```
----jGRASP exec: python Selection03.py
Enter Sales  -->  300000
Christmas bonus: 0
----jGRASP: operation complete.
```

structure
mmands as
ion is used.

```
7  print()
8  sales = eval(input("Enter Sales  -->  "))
9  bonus = 0
10
11 if sales >= 500000:
12    print("\nCONGRATULATIONS!")
13    print("You sold half a million dollars in merchandise!")
14    print("You will receive a $1000 Christmas Bonus!")
15    print("Keep up the good work!")
16    bonus = 1000
17
18 print()
19 print("Christmas bonus:",bonus)
20
```

```
----jGRASP exec: python
Enter Sales  -->  300000
Christmas bonus: 0
----jGRASP: operation co
```

```
----jGRASP exec: python Selection03.py
Enter Sales  -->  500000

CONGRATULATIONS!
You sold half a million dollars in merchandise!
You will receive a $1000 Christmas Bonus!
Keep up the good work!

Christmas bonus: 1000

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

```
 7 print()
 8 sales = eval(input("En
 9 bonus = 0
10
11 if sales >= 500000:
12     print("\nCONGRATULATIONS!")
13     print("You sold half a million dollars in merchandise!")
14     print("You will receive a $1000 Christmas Bonus!")
15     print("Keep up the good work!")
16     bonus = 1000
17
18 print()
19 print("Christmas bonus:",bonus)
20
```

One-Way Selection

General Syntax:

if *condition is True*:
    execute program statement(s)

Specific Examples:

```
if gpa >= 90:
    print("Honor Grad!")


if savings >= 10000:
    print("It's skiing time")
    print("Let's pack")
    print("Remember your skis")
```

You can control as many statements as you wish with a control structure, as long as you use proper, consistent indentation.