

# **Exposure CS 2021** **for CS1**

## **Chapter 7 Section 6-10 Slides**

**2-Way, Multi-Way & Nested Selection  
and Formatting Numerical Output**

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



# Section 7.6

## Two Way Selection

# Two-Way Selection

## Real Life Example



**I35W takes you  
to Fort Worth.**

**I35E takes you  
to Dallas.**

**Interstate 35 splits into I35W and I35E just North of Hillsboro.**

```
1 # Selection04.py
2 # This program demonstrates two-way selection
3 # with <if..else>. Run the program twice:
4 # First with 1200, then with 1000.
5
6
7 print()
8 sat = eval(input("Enter SAT score --> "))
9 print()
10
11 if sat >= 1100:
12     print("You are admitted.")
13 else:
14     print("You are not admitted.")
15
```



```
----jGRASP exec: python Se
```

```
Enter SAT score --> 1200
```

```
You are admitted.
```

```
----jGRASP: operation comp
```

illustrates two-way selection  
in the program twice:  
with 1000.

```
7 print()  
8 sat = eval(input("Enter SAT score --> "))  
9 print()  
10  
11 if sat >= 1100:  
12     print("You are admitted.")  
13 else:  
14     print("You are not admitted.")  
15
```

```
----jGRASP exec: python Se
```

```
Enter SAT score --> 1200
```

```
You are admitted.
```

```
----jGRASP: operation comp
```

```
----jGRASP exec: python Se
```

```
Enter SAT score --> 1000
```

```
You are not admitted.
```

```
----jGRASP: operation comp
```

```
7 print()  
8 sat = eval(input("Enter SAT score --> "))  
9 print()  
10  
11 if sat >= 1100:  
12     print("You are admitted.")  
13 else:  
14     print("You are not admitted.")  
15
```

```
1 # Selection05.py
2 # This program demonstrates that multiple program
3 # statements can be controlled in both parts of an
4 # <if..else> structure as long as proper, consistent
5 # indentation is used. Run the program twice:
6 # First with 1100, then with 1099.
7
8
9 print()
10 sat = eval(input("Enter SAT score --> "))
11 print()
12
13 if sat >= 1100:
14     print("You are admitted.")
15     print("Orientation will start in June.")
16 else:
17     print("You are not admitted.")
18     print("Please try again when your SAT improves.")
19
```

```
----jGRASP exec: python Select
```

```
Enter SAT score --> 1100
```

```
You are admitted.
```

```
Orientation will start in June.
```

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

```
----jGRASP exec: python Selection05.py
```

```
Enter SAT score --> 1099
```

```
You are not admitted.
```

```
Please try again when your SAT improves.
```

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

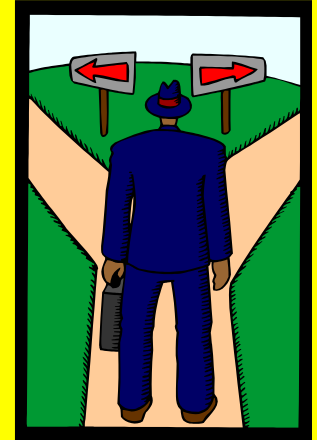
```
9 print()
10 sat = eval(input("Enter SAT score --> "))
11 print()
12
13 if sat >= 1100:
14     print("You are admitted.")
15     print("Orientation will start in June.")
16 else:
17     print("You are not admitted.")
18     print("Please try again when your SAT improves.")
19
```



# Two-Way Selection

## General Syntax:

```
if condition is True:  
    execute program statement(s)  
else: # when condition is False  
    execute alternate program statement(s)
```



## Specific Example:

```
if average >= 70:  
    print("You passed!")  
    print("Get ready for summer vacation!")  
else:  
    print("You failed.")  
    print("Get ready for summer school.")
```



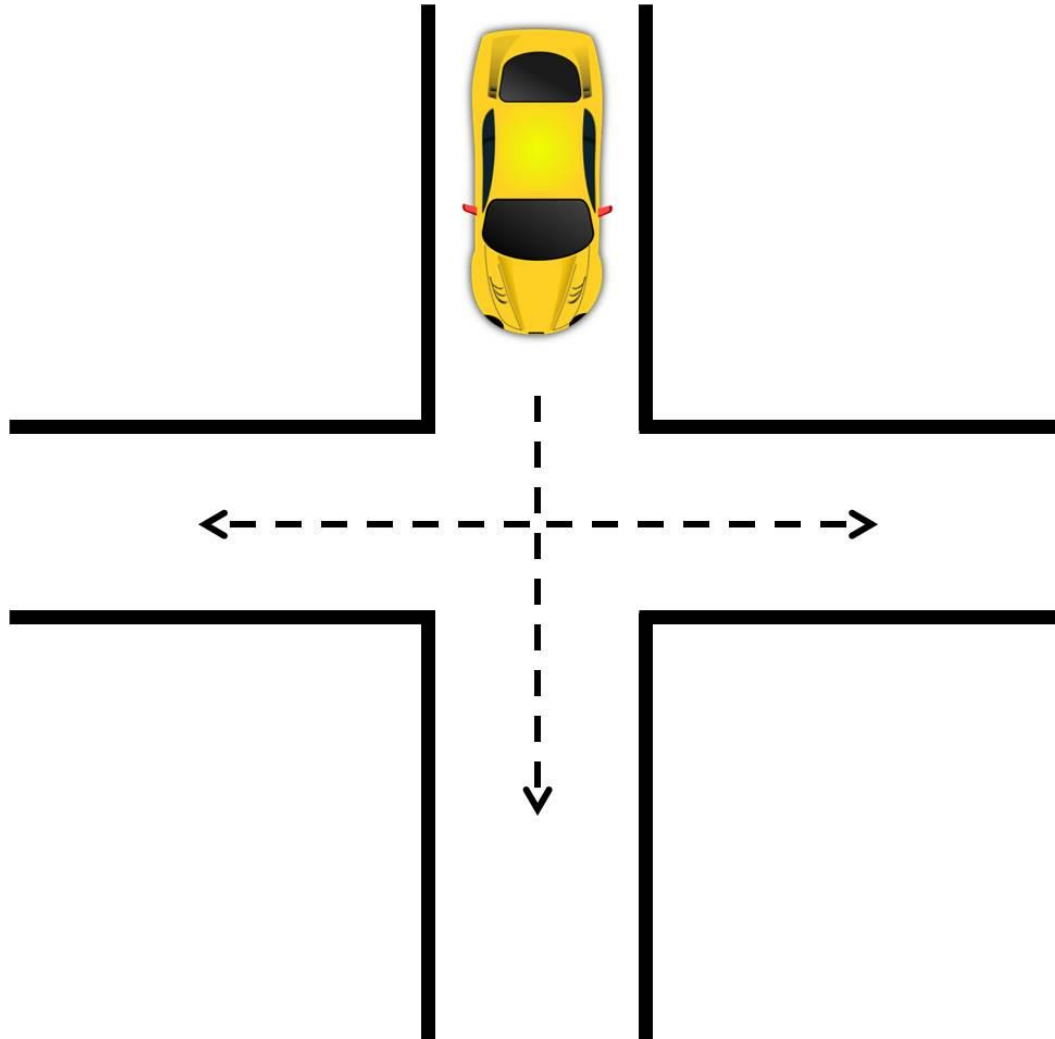
# Section 7.7

# Multi-Way Selection

A decorative graphic on the right side of the slide. It features a vertical orange bar. To the right of the bar, there are several concentric yellow circles. Overlaid on this are several arrows: a thick black arrow pointing upwards and to the right, a thinner black arrow pointing upwards and to the left, and a white arrow pointing upwards and to the right. The background of the slide is a gradient from dark blue at the top to red at the bottom.

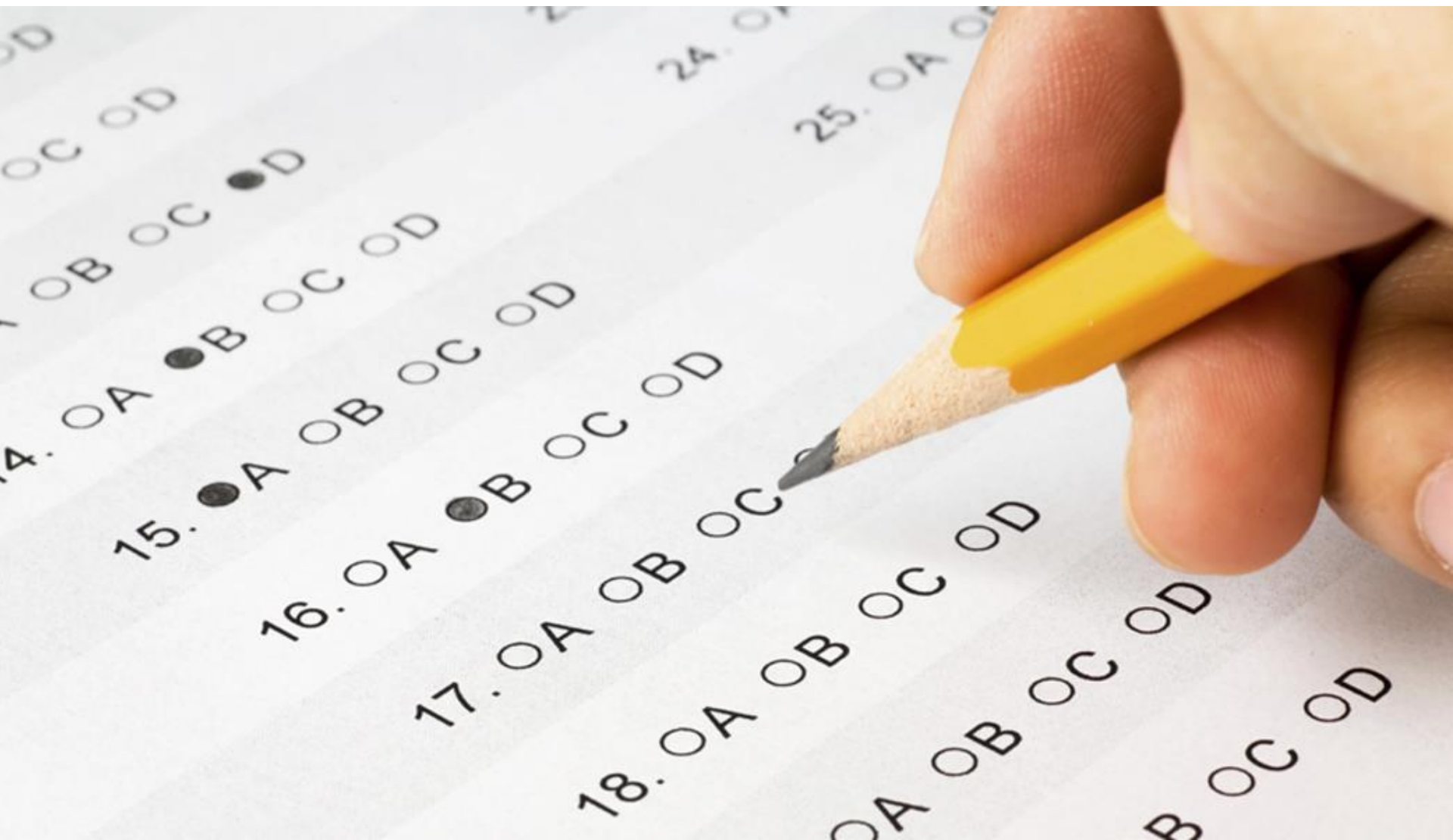
# Multi-Way Selection

## Real Life Example #1



# Multi-Way Selection

## Real Life Example #2





# Multi-Way Selection

## Real Life Example #3



Pistachio

Sweet Cream

Mocha

Salted Caramel

Chocolate

Cotton Candy

French Vanilla

Butter Pecan

```
1 # Selection06.py
2 # This program is supposed to display the letter grade
3 # earned based on the number grade entered by the user.
4 # Since there are more than 2 possible paths (A,B,C,D,F)
5 # this would be an example of "Multi-Way Selection" if
6 # the program worked. Using 5 separate <if> statements
7 # has created a Logic Error that gives strange output.
8
9
10 print()
11 grade = eval(input("Enter Number Grade --> "))
12 print()
13
14 if grade >= 90:
15     print("You earned an A!")
16 if grade >= 80:
17     print("You earned a B.")
18 if grade >= 70:
19     print("You earned a C.")
20 if grade >= 60:
21     print("You earned a D.")
22 if grade >= 0:
23     print("You earned an F.")
```



```
1 # Selection06.py
2 # This program is supposed to
3 # earned based on the number
4 # Since there are more than 2
5 # this would be an example of
6 # the program worked. Using !
7 # has created a Logic Error th
```

```
8
9
10 print()
11 grade = eval(input("Enter Number Grade --> "))
12 print()
13
14 if grade >= 90:
15     print("You earned an A!")
16 if grade >= 80:
17     print("You earned a B.")
18 if grade >= 70:
19     print("You earned a C.")
20 if grade >= 60:
21     print("You earned a D.")
22 if grade >= 0:
23     print("You earned an F.")
```

```
----jGRASP exec: python Sele
```

```
Enter Number Grade --> 50
```

```
You earned an F.
```

```
----jGRASP: operation comple
```

```

1 # Selection06.py
2 # This program is supposed to
3 # earned based on the number
4 # Since there are more than 2
5 # this would be an example of
6 # the program worked. Using !
7 # has created a Logic Error th
8
9
10 print()
11 grade = eval(input("Enter Number Grade --> "))
12 print()
13
14 if grade >= 90:
15     print("You earned an A")
16 if grade >= 80:
17     print("You earned a B.")
18 if grade >= 70:
19     print("You earned a C.")
20 if grade >= 60:
21     print("You earned a D.")
22 if grade >= 0:
23     print("You earned an F")

```

```

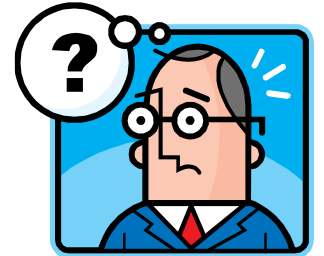
----jGRASP exec: python Sele
Enter Number Grade --> 50
You earned an F.
----jGRASP: operation comple

```

```

----jGRASP exec: python Sele
Enter Number Grade --> 100
You earned an A!
You earned a B.
You earned a C.
You earned a D.
You earned an F.
----jGRASP: operation comple

```



```
1 # Selection07.py
2 # This program fixes the Logic Error of the
3 # previous program by adding several strategic
4 # <else> statements which will ensure that only
5 # 1 letter grade is displayed. While this works,
6 # the program's indentation is somewhat annoying.
7
8
9 print()
10 grade = eval(input("Enter Number Grade --> "))
11 print()
12
13 if grade >= 90:
14     print("You earned an A!")
15 else:
16     if grade >= 80:
17         print("You earned a B.")
18     else:
19         if grade >= 70:
20             print("You earned a C.")
21         else:
22             if grade >= 60:
23                 print("You earned a D.")
24             else:
25                 print("You earned an F.")
```

```
----jGRASP exec: python Sele
```

```
Enter Number Grade --> 100
```

```
You earned an A!
```

```
----jGRASP: operation comple
```

```
----jGRASP exec: python Sele
```

```
Enter Number Grade --> 65
```

```
You earned a D.
```

```
----jGRASP: operation comple
```

```
13 if grade >= 90:
14     print("You earned an A!")
15 else:
16     if grade >= 80:
17         print("You earned a B.")
18     else:
19         if grade >= 70:
20             print("You earned a C.")
21         else:
22             if grade >= 60:
23                 print("You earned a D.")
24             else:
25                 print("You earned an F.")
```

```
1 # Selection08.py
2 # This program shows a better way to do "Multi-Way
3 # Selection" using <if..elif..else>. The <elif>
4 # command essentially combines the <else> with the
5 # next <if>. Not only is this less code to type,
6 # it also has nicer indentation."
7
8
9 print()
10 grade = eval(input("Enter Number Grade --> "))
11 print()
12
13 if grade >= 90:
14     print("You earned an A!")
15 elif grade >= 80:
16     print("You earned a B.")
17 elif grade >= 70:
18     print("You earned a C.")
19 elif grade >= 60:
20     print("You earned a D.")
21 else:
22     print("You earned an F.")
```

```
----jGRASP exec: python Sel
```

```
Enter Number Grade --> 85
```

```
You earned a B.
```

```
----jGRASP: operation compl
```

```
----jGRASP exec: python Sel
```

```
Enter Number Grade --> 75
```

```
You earned a C.
```

```
----jGRASP: operation compl
```

```
12
13 if grade >= 90:
14     print("You earned an A!")
15 elif grade >= 80:
16     print("You earned a B.")
17 elif grade >= 70:
18     print("You earned a C.")
19 elif grade >= 60:
20     print("You earned a D.")
21 else:
22     print("You earned an F.")
```



```
1 # Selection09.py
2 # This program demonstrates a number of things:
3 # 1. Selection can be based on text values also,
4 #    not just number values.
5 # 2. As with other selection structures, Multi-Way
6 #    Selection can control multiple programming
7 #    commands as long as proper, consistent
8 #    indentation is used.
9 # 3. The program will not work properly if the
10 #    user does not enter an A, B, C, D or F.
11
12
13 print()
14 grade = input("Enter Letter Grade --> ")
15 print()
16
17 if grade == 'A':
18     print("You grade is 90 or above.")
19     print("Excellent!")
20 elif grade == 'B':
21     print("You grade is in the 80s.")
22     print("Good")
23 elif grade == 'C':
24     print("You grade is in the 70s.")
25     print("Fair")
26 elif grade == 'D':
27     print("You grade is in the 60s.")
28     print("Poor")
29 elif grade == 'F':
30     print("You grade is below 60.")
31     print("Bad")
```

```

1 # Selection09.py
2 # This program demonstrates a number of things
3 # 1. Selection can be based on text values as well as
4 #    not just number values.
5 # 2. As with other selection structures, Multiple
6 #    Selection can control multiple programmatic
7 #    commands as long as proper, consistent
8 #    indentation is used.
9 # 3. The program will not work properly if the
10 #    user does not enter an A, B, C, D or F.
11
12
13 print()
14 grade = input("Enter Letter Grade --> ")
15 print()
16
17 if grade == 'A':
18     print("You grade is 90 or above.")
19     print("Excellent!")
20 elif grade == 'B':
21     print("You grade is in the 80s.")
22     print("Good")
23 elif grade == 'C':
24     print("You grade is in the 70s.")
25     print("Fair")
26 elif grade == 'D':
27     print("You grade is in the 60s.")
28     print("Poor")
29 elif grade == 'F':
30     print("You grade is below 60.")
31     print("Bad")

```

```
----jGRASP exec: python Se
```

```
Enter Letter Grade --> A
```

```
You grade is 90 or above.
Excellent!
```

```
----jGRASP: operation comp
```

```
----jGRASP exec: python Se
```

```
Enter Letter Grade --> B
```

```
You grade is in the 80s.
Good
```

```
----jGRASP: operation comp
```

```
----jGRASP exec: python Se
```

```
Enter Letter Grade --> Q
```

```
----jGRASP: operation comp
```

```
1 # Selection10.py
2 # This program demonstrates how the <else> command
3 # is used in Multi-Way Selection to deal with the
4 # case of a value that does not match any of the
5 # cases in your <if..elif> structure.
6
7
8 print()
9 grade = input("Enter Letter Grade --> ")
10 print()
11
12 if grade == 'A':
13     print("You grade is 90 or above.")
14     print("Excellent!")
15 elif grade == 'B':
16     print("You grade is in the 80s.")
17     print("Good")
18 elif grade == 'C':
19     print("You grade is in the 70s.")
20     print("Fair")
21 elif grade == 'D':
22     print("You grade is in the 60s.")
23     print("Poor")
24 elif grade == 'F':
25     print("You grade is below 60.")
26     print("Bad")
27 else:
28     print("You did not enter an A, B, C, D or F.")
29     print("Please re-run the program and try again.")
```

```
1 # Selection10.py
2 # This program de
3 # is used in Mult
4 # case of a value
5 # cases in your
6
7
8 print()
9 grade = input("Enter Letter Grade: ")
10 print()
11
12 if grade == 'A':
13     print("You grade is 90 or above.")
14     print("Excellent!")
15 elif grade == 'B':
16     print("You grade is in the 80s.")
17     print("Good")
18 elif grade == 'C':
19     print("You grade is in the 70s.")
20     print("Fair")
21 elif grade == 'D':
22     print("You grade is in the 60s.")
23     print("Poor")
24 elif grade == 'F':
25     print("You grade is below 60.")
26     print("Bad")
27 else:
28     print("You did not enter an A, B, C, D or F.")
29     print("Please re-run the program and try again.")
```

```
----jGRASP exec: python Selection10.py
```

```
Enter Letter Grade --> Q
```

```
You did not enter an A, B, C, D or F.
Please re-run the program and try again.
```

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

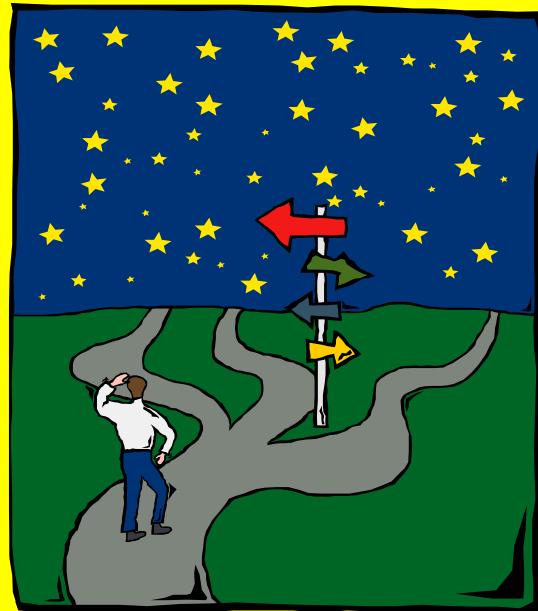
# Multi-Way Selection

## General Syntax:

```
if first condition is True:  
    execute first set of program statements  
elif second condition is True:  
    execute second set of program statements  
:  
:  
:  
:  
:  
:  
else: # when all above conditions are False  
    execute default set of program statements
```

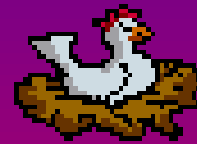
## Specific Example:

```
if grade == 'A':  
    points = 4.0  
elif grade == 'B':  
    points = 3.0  
elif grade == 'C':  
    points = 2.0  
elif grade == 'D':  
    points = 1.0  
elif grade == 'F':  
    points = 0.0  
else:  
    print("Error. Please try again.")
```

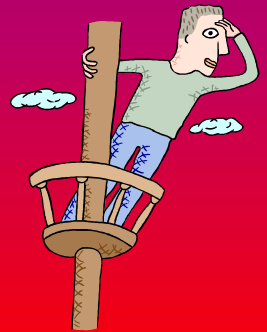


# Section 7.8

# Nested



# Selection





```
1 # Selection11.py
2 # This program has 2 separate <if..else> structures.
3 # The first determines if a student is admitted to
4 # the college based on his/her SAT score.
5 # The second determines if that student qualifies
6 # for financial aid based on his/her family income.
7 # The problem with this program is that even if the
8 # student is not admitted, it still asks about
9 # family income and has the potential of telling
10 # a student who was not admitted that he/she
11 # qualifies for financial aid.
12
13
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 else:
22     print("You are not admitted.")
23     print("Please try again when your SAT improves.")
24
25
26 print()
27 income = eval(input("Enter your family income --> "))
28 print()
29
30 if income < 20000:
31     print("You qualify for financial aid.")
32 else:
33     print("You do not qualify for financial aid.")
```

```

1 # Selection11.py
2 # This program has 2 separate <if..
3 # The first determines if a student
4 # the college based on his/her SAT
5 # The second determines is that stu
6 # for financial aid based on his/he
7 # The problem with this program is
8 # student is not admitted, it still
9 # family income and has the potenti
10 # a student who was not admitted th
11 # qualifies for financial aid.
12
13
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
21 else:
22     print("You are not admitted.")
23     print("Please try again when you
24
25
26 print()
27 income = eval(input("Enter your famil
28 print()
29
30 if income < 20000:
31     print("You qualify for financial
32 else:
33     print("You do not qualify for fi

```

```
----jGRASP exec: python Selection11.py
```

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 Selection11.py
```

Enter SAT score --> 1000

You are not admitted.

Please try again when your SAT improves.

Enter your family income --> 19000

You qualify for financial aid.

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

```
1 # Selection12.py
2 # This program fixes the issue of the previous program
3 # by "nesting" the second <if..else> structure inside
4 # the <if> part of the first. Now, the "family income"
5 # question is only asked to students who are admitted.
6 # NOTE: Proper indentation is VERY important here.
7
8
9 print()
10 sat = eval(input("Enter SAT score --> "))
11 print()
12
13 if sat >= 1100:
14     print("You are admitted.")
15     print("Orientation will start in June.")
16     print()
17     income = eval(input("Enter your family income --> "))
18     print()
19     if income < 20000:
20         print("You qualify for financial aid.")
21     else:
22         print("You do not qualify for financial aid.")
23 else:
24     print("You are not admitted.")
25     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 complete.
```

```
----jGRASP exec: python Selection12.py
```

▶▶ Enter SAT score --> 700

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

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

```
----jGRASP exec: python Selection12.py
```

▶▶ 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.
```

# Section 7.9



combining

Selection

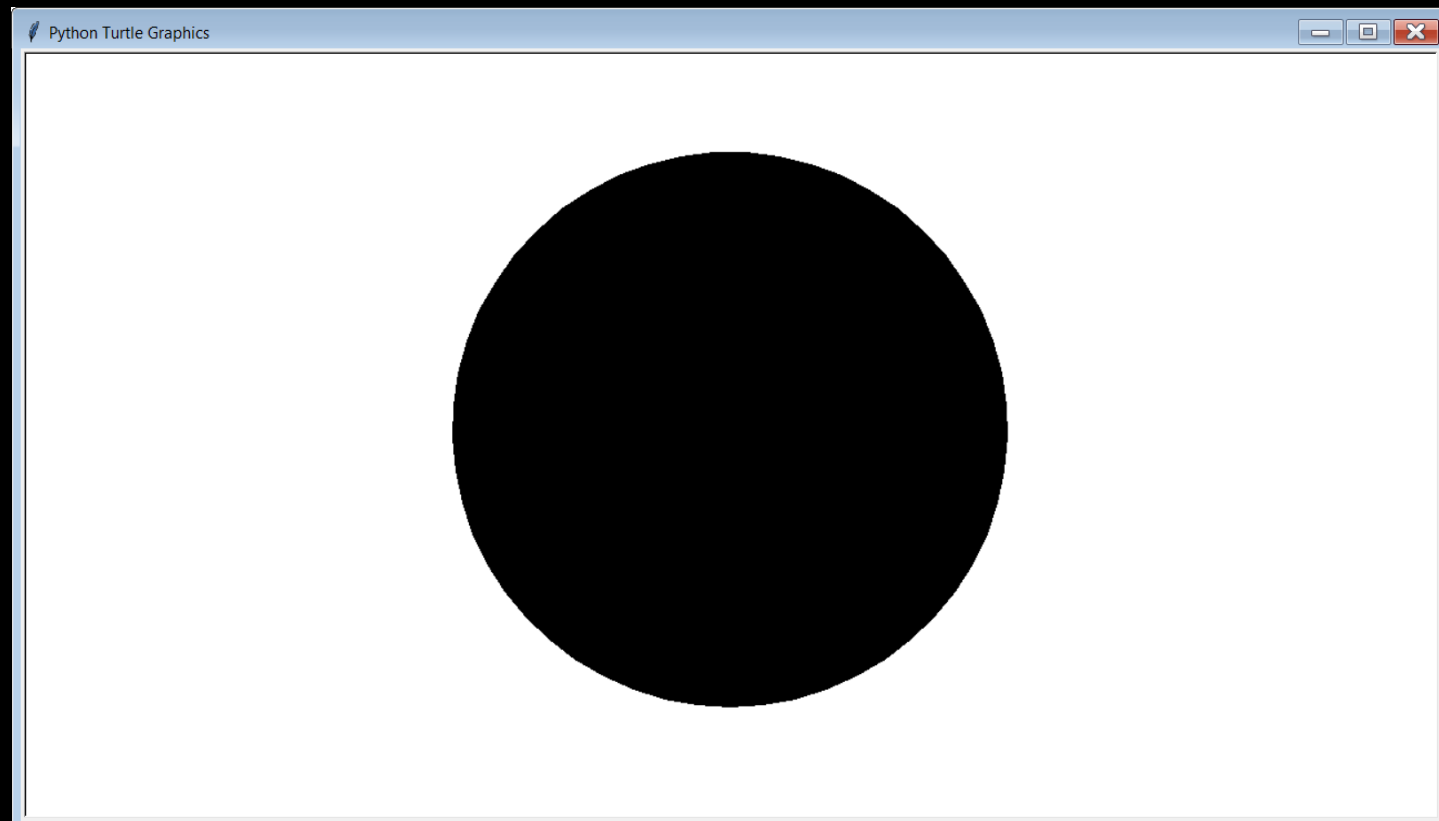
with Graphics

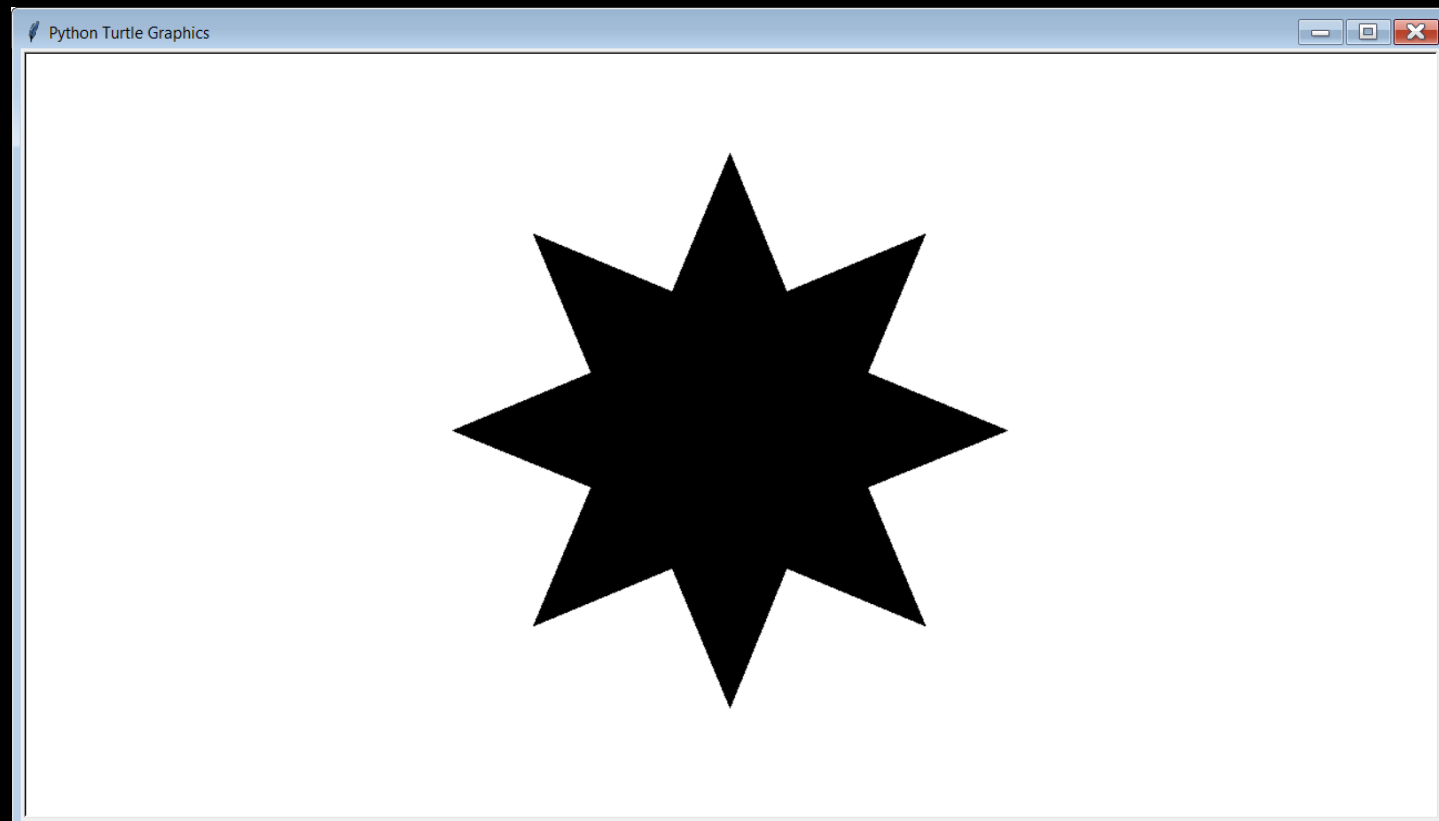
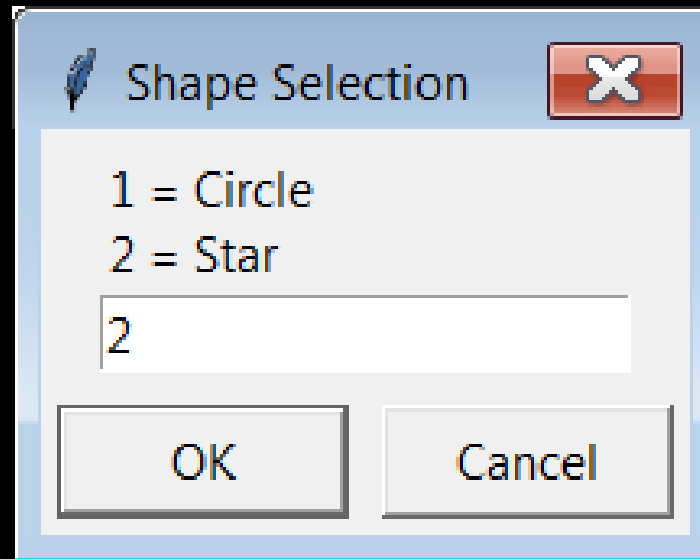
```
1 # Selection13.py
2 # This program demonstrates that selection can be used
3 # to manipulate the output of a graphics program.
4 # This is the very thing you will be doing in Lab 7B.
5
6
7 from Graphics import *
8
9 beginGrfx(1300,700)
10
11 shapeNum = numinput("Shape Selection","1 = Circle \n2 = Star")
12
13 if shapeNum == 1:
14     fillCircle(650,350,250)
15 elif shapeNum == 2:
16     fillStar(650,350,250,8)
17 else:
18     drawString("You did not enter a 1 or a 2.",100,315,"Arial",48,"bold")
19     drawString("Please try again.",100,465,"Arial",48,"bold")
20
21 endGrfx()
```

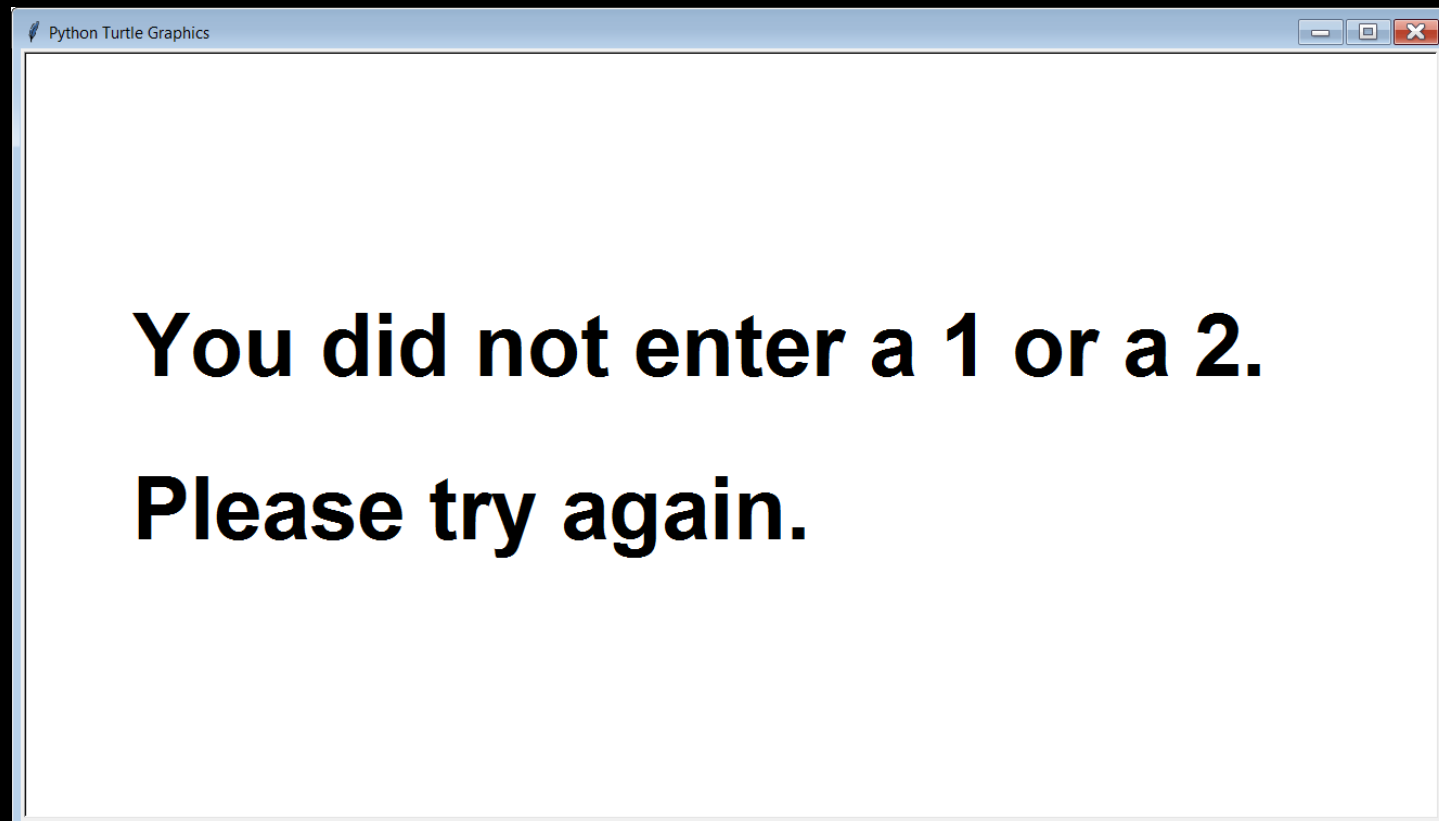
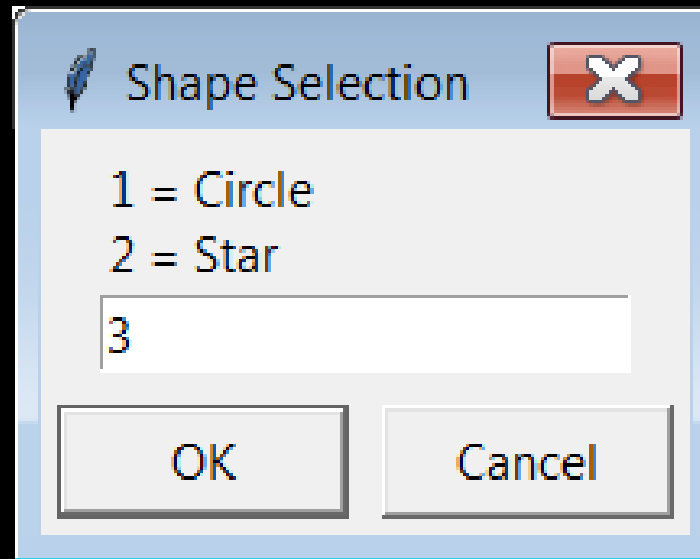


 Shape Selection 

1 = Circle  
2 = Star









## Shape Selection



1 = Circle

2 = Star

Q

OK

Cancel



## Illegal value



Not a floating point value.  
Please try again

OK

# Section 7.10

Formatting

Numerical Output

```
1 # NumberFormat01.py
2 # This program demonstrates that by default
3 # numbers are displayed "left-justified"
4 # which means they do not line up by their
5 # place value.
6
7
8 print()
9 print(1)
10 print(12)
11 print(123)
12 print(1234)
13 print(12345)
14
```

```
-----jGRASP
1
12
123
1234
12345
-----jGRASP:
```

```
1 # NumberFormat02.py
2 # This program demonstrates one way to properly line
3 # numbers up by place value using the <format> command.
4 # The 05 inside "{:05}" means each number will have
5 # enough 0s placed at the front of the number to force
6 # it to be displayed as a 5 digit number.
7
8
9 print()
10 print("{:05}".format(1))
11 print("{:05}".format(12))
12 print("{:05}".format(123))
13 print("{:05}".format(1234))
14 print("{:05}".format(12345))
15
```

-----jGRASP

00001

00012

00123

01234

12345

-----jGRASP:



```
1 # NumberFormat03.py
2 # This program demonstrates that leading 0s are
3 # not required to line up numbers by place value.
4 # Just leave out the 0, and the number will be
5 # displayed with leading spaces instead.
6
7
8 print()
9 print("{:5}".format(1))
10 print("{:5}".format(12))
11 print("{:5}".format(123))
12 print("{:5}".format(1234))
13 print("{:5}".format(12345))
14
```

```
-----jGRASP
      1
     12
    123
   1234
  12345
-----jGRASP:
```

```
1 # NumberFormat04.py
2 # This program demonstrates what happens when the
3 # format size for the number is not large enough.
4 # If the format is not possible, it is simply
5 # ignored.  There is no error message.
6
7
8 print()
9 print("{:5}".format(1))
10 print("{:5}".format(12))
11 print("{:5}".format(123))
12 print("{:5}".format(1234))
13 print("{:5}".format(12345))
14 print("{:5}".format(123456))
15 print("{:5}".format(1234567))
16 print("{:5}".format(12345678))
17 print("{:5}".format(123456789))
```

```
1 # NumberFormat04.py
2 # This program demonstrates what happens when the
3 # format size for the number is not large enough.
4 # If the format is not possible, it is simply
5 # ignored. There is no error message.
6
7
8 print()
9 print("{:5}".format(1))
10 print("{:5}".format(12))
11 print("{:5}".format(123))
12 print("{:5}".format(1234))
13 print("{:5}".format(12345))
14 print("{:5}".format(123456))
15 print("{:5}".format(1234567))
16 print("{:5}".format(12345678))
17 print("{:5}".format(123456789))
```

----jGRASP

1

12

123

1234

12345

123456

1234567

12345678

123456789

----jGRASP:

# **format Command Reality**

While the **format** command can change the appearance of a number, it cannot change the value of a number.

This is why it is OK to add leading zeros or spaces when a number does not have enough digits. It is also why it is not OK to remove digits when a number has too many.

```
1 # NumberFormat05.py
2 # This program demonstrates how to add commas (,)
3 # as a "thousand separator". It may seem like
4 # it stops working at 10 million. The problem is
5 # the commas count as digits.
6
7
8 print()
9 print("{:9,}".format(1))
10 print("{:9,}".format(12))
11 print("{:9,}".format(123))
12 print("{:9,}".format(1234))
13 print("{:9,}".format(12345))
14 print("{:9,}".format(123456))
15 print("{:9,}".format(1234567))
16 print("{:9,}".format(12345678))
17 print("{:9,}".format(123456789))
```

```
1 # NumberFormat05.py
2 # This program demonstrates how to add commas (,)
3 # as a "thousand separator". It may seem like
4 # it stops working at 10 million. The problem is
5 # the commas count as digits.
6
7
8 print()
9 print("{:9,}".format(1))
10 print("{:9,}".format(12))
11 print("{:9,}".format(123))
12 print("{:9,}".format(1234))
13 print("{:9,}".format(12345))
14 print("{:9,}".format(123456))
15 print("{:9,}".format(1234567))
16 print("{:9,}".format(12345678))
17 print("{:9,}".format(123456789))
```

----jGRASP

```

1
12
123
1,234
12,345
123,456
1,234,567
12,345,678
123,456,789
```

----jGRASP:

# The problem with program NumberFormat05.py

Number	1	2	3	,	4	5	6	,	7	8	9
Count	1	2	3	4	5	6	7	8	9	10	11

The number in **format**'s string literal does not specify the total number of digits, it specifies the total number of characters which includes digits, commas, and even the decimal point.





```
1 # NumberFormat07.py
2 # This program demonstrates that commas can
3 # be used without leading zeros or spaces.
4
5
6 print()
7 print("{:,}".format(1))
8 print("{:,}".format(12))
9 print("{:,}".format(123))
10 print("{:,}".format(1234))
11 print("{:,}".format(12345))
12 print("{:,}".format(123456))
13 print("{:,}".format(1234567))
14 print("{:,}".format(12345678))
15 print("{:,}".format(123456789))
```

----jGRASP

```
1
12
123
1,234
12,345
123,456
1,234,567
12,345,678
123,456,789
```

----jGRASP:

```
1 # NumberFormat08.py
2 # This program repeats Documentation02.py from Chapter 3.
3 # One problem with the program still has is that the
4 # numbers which are displayed represent money, but they
5 # are not rounded to 2 decimal places.
6
7
8 hoursWorked = 35
9 hourlyRate = 8.75
10 grossPay = hoursWorked * hourlyRate
11 deductions = grossPay * 0.29
12 netPay = grossPay - deductions
13
14 print()
15 print("Hours Worked: ",hoursWorked)
16 print("Hourly Rate:  ",hourlyRate)
17 print("Gross Pay:    ",grossPay)
18 print("Deductions:   ",deductions)
19 print("Net Pay:      ",netPay)
```

```
1 # NumberFormat08.py
2 # This program repeats D
3 # One problem with the p
4 # numbers which are disp
5 # are not rounded to 2 d
6
7
8 hoursWorked = 35
9 hourlyRate = 8.75
10 grossPay = hoursWorked * hourlyRate
11 deductions = grossPay * 0.29
12 netPay = grossPay - deductions
13
14 print()
15 print("Hours Worked: ",hoursWorked)
16 print("Hourly Rate:   ",hourlyRate)
17 print("Gross Pay:      ",grossPay)
18 print("Deductions:     ",deductions)
19 print("Net Pay:         ",netPay)
```

```
-----jGRASP exec: python

Hours Worked:      35
Hourly Rate:       8.75
Gross Pay:         306.25
Deductions:        88.8125
Net Pay:           217.4375

-----jGRASP: operation c
```

```
1 # NumberFormat09.py
2 # This program demonstrates how to format real number
3 # output with the <format> command. The f inside
4 # "{:6.2f}" means this is a "floating-point number"
5 # which is the same thing as a "real number".
6 # The 6 indicates the entire number will be 6 characters
7 # long, including the decimal point. The 2 indicates
8 # there will be 2 digits after the decimal point.
9
10
11 hoursWorked = 35
12 hourlyRate = 8.75
13 grossPay = hoursWorked * hourlyRate
14 deductions = grossPay * 0.29
15 netPay = grossPay - deductions
16
17 print()
18 print("Hours Worked:", "{:6.2f}".format(hoursWorked))
19 print("Hourly Rate: ", "{:6.2f}".format(hourlyRate))
20 print("Gross Pay:    ", "{:6.2f}".format(grossPay))
21 print("Deductions:   ", "{:6.2f}".format(deductions))
22 print("Net Pay:      ", "{:6.2f}".format(netPay))
```

```
1 # NumberFormat09.py
2 # This program demonstrates
3 # output with the <format>
4 # "{:6.2f}" means this is a
5 # which is the same thing as
6 # The 6 indicates the entire
7 # long, including the decimal
8 # there will be 2 digits after
9
10
11 hoursWorked = 35
12 hourlyRate = 8.75
13 grossPay = hoursWorked * hourlyRate
14 deductions = grossPay * 0.29
15 netPay = grossPay - deductions
16
17 print()
18 print("Hours Worked:", "{:6.2f}".format(hoursWorked))
19 print("Hourly Rate: ", "{:6.2f}".format(hourlyRate))
20 print("Gross Pay:    ", "{:6.2f}".format(grossPay))
21 print("Deductions:   ", "{:6.2f}".format(deductions))
22 print("Net Pay:      ", "{:6.2f}".format(netPay))
```

```
----jGRASP exec: python
```

```
Hours Worked:    35.00
Hourly Rate:      8.75
Gross Pay:        306.25
Deductions:       88.81
Net Pay:          217.44
```

```
----jGRASP: operation c
```

```
1 # NumberFormat10.py
2 # This program demonstrates how to format
3 # real numbers without leading spaces.
4
5
6 hoursWorked = 35
7 hourlyRate = 8.75
8 grossPay = hoursWorked * hourlyRate
9 deductions = grossPay * 0.29
10 netPay = grossPay - deductions
11
12 print()
13 print("Hours Worked:", "{:.2f}".format(hoursWorked))
14 print("Hourly Rate: ", "{:.2f}".format(hourlyRate))
15 print("Gross Pay:    ", "{:.2f}".format(grossPay))
16 print("Deductions:   ", "{:.2f}".format(deductions))
17 print("Net Pay:      ", "{:.2f}".format(netPay))
```



```
1 # NumberFormat10.py
2 # This program demonstra
3 # real numbers without l
4
5
6 hoursWorked = 35
7 hourlyRate = 8.75
8 grossPay = hoursWorked *
9 deductions = grossPay *
10 netPay = grossPay - deductions
11
12 print()
13 print("Hours Worked:", "{:.2f}".format(hoursWorked))
14 print("Hourly Rate: ", "{:.2f}".format(hourlyRate))
15 print("Gross Pay:    ", "{:.2f}".format(grossPay))
16 print("Deductions:   ", "{:.2f}".format(deductions))
17 print("Net Pay:      ", "{:.2f}".format(netPay))
```

```
----jGRASP exec: python
```

```
Hours Worked: 35.00
Hourly Rate:  8.75
Gross Pay:    306.25
Deductions:   88.81
Net Pay:      217.44
```

```
----jGRASP: operation c
```

```
1 # NumberFormat11.py
2 # This program demonstrates that real numbers can be
3 # formatted with leading spaces, rounded digits past
4 # the decimal point, commas, and even dollar sign$.
5
6 hoursWorked = 50
7 hourlyRate = 199.98
8 grossPay = hoursWorked * hourlyRate
9 deductions = grossPay * 0.29
10 netPay = grossPay - deductions
11
12 print()
13 print("Hours Worked:",hoursWorked)
14 print("Hourly Rate: ", "${:8,.2f}".format(hourlyRate))
15 print("Gross Pay:    ", "${:8,.2f}".format(grossPay))
16 print("Deductions:   ", "${:8,.2f}".format(deductions))
17 print("Net Pay:      ", "${:8,.2f}".format(netPay))
```

```
1 # NumberFormat11.py
2 # This program demo
3 # formatted with le
4 # the decimal point
5
6 hoursWorked = 50
7 hourlyRate = 199.98
8 grossPay = hoursWor
9 deductions = grossP
10 netPay = grossPay -
11
12 print()
13 print("Hours Worked:", hoursWorked)
14 print("Hourly Rate: ", "${:8,.2f}".format(hourlyRate))
15 print("Gross Pay:    ", "${:8,.2f}".format(grossPay))
16 print("Deductions:   ", "${:8,.2f}".format(deductions))
17 print("Net Pay:      ", "${:8,.2f}".format(netPay))
```

```
----jGRASP exec: python
```

```
Hours Worked: 50
Hourly Rate:   $ 199.98
Gross Pay:     $9,999.00
Deductions:    $2,899.71
Net Pay:       $7,099.29
```

```
----jGRASP: operation c
```