Exposure CS 2022 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.

```python
# Selection04.py
# This program demonstrates two-way selection
# with <if..else>.  Run the program twice:
# First with 1200, then with 1000.


print()
sat = eval(input("Enter SAT score  -->  "))
print()

if sat >= 1100:
    print("You are admitted.")
else:
    print("You are not admitted.")

```

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

Enter SAT score  -->  1200

You are admitted.

----jGRASP: operation com
```

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

Enter SAT score  -->  1000

You are not admitted.

----jGRASP: operation com
```

```python
print()
sat = eval(input("Enter SAT score  -->  "))
print()

if sat >= 1100:
    print("You are admitted.")
else:
    print("You are not admitted.")

```

```python
# Selection05.py
# This program demonstrates that multiple program
# statements can be controlled in both parts of an
# <if...else> structure as long as proper, consistent
# indentation is used.  Run the program twice:
# First with 1100, then with 1099.


print()
sat = eval(input("Enter SAT score  -->  "))
print()

if sat >= 1100:
   print("You are admitted.")
   print("Orientation will start in June.")
else:
   print("You are not admitted.")
   print("Please try again when your SAT improves.")
```

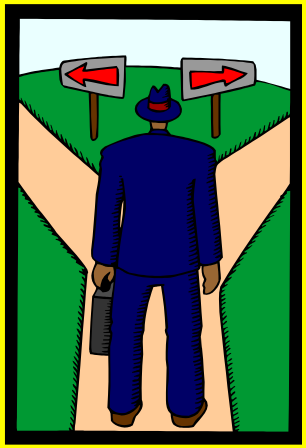
```
----jGRASP exec: python Select

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.
```

```python

print()
sat = eval(input("Enter SAT score  -->  "))
print()

if sat >= 1100:
    print("You are admitted.")
    print("Orientation will start in June.")
else:
    print("You are not admitted.")
    print("Please try again when your SAT improves.")
```

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:

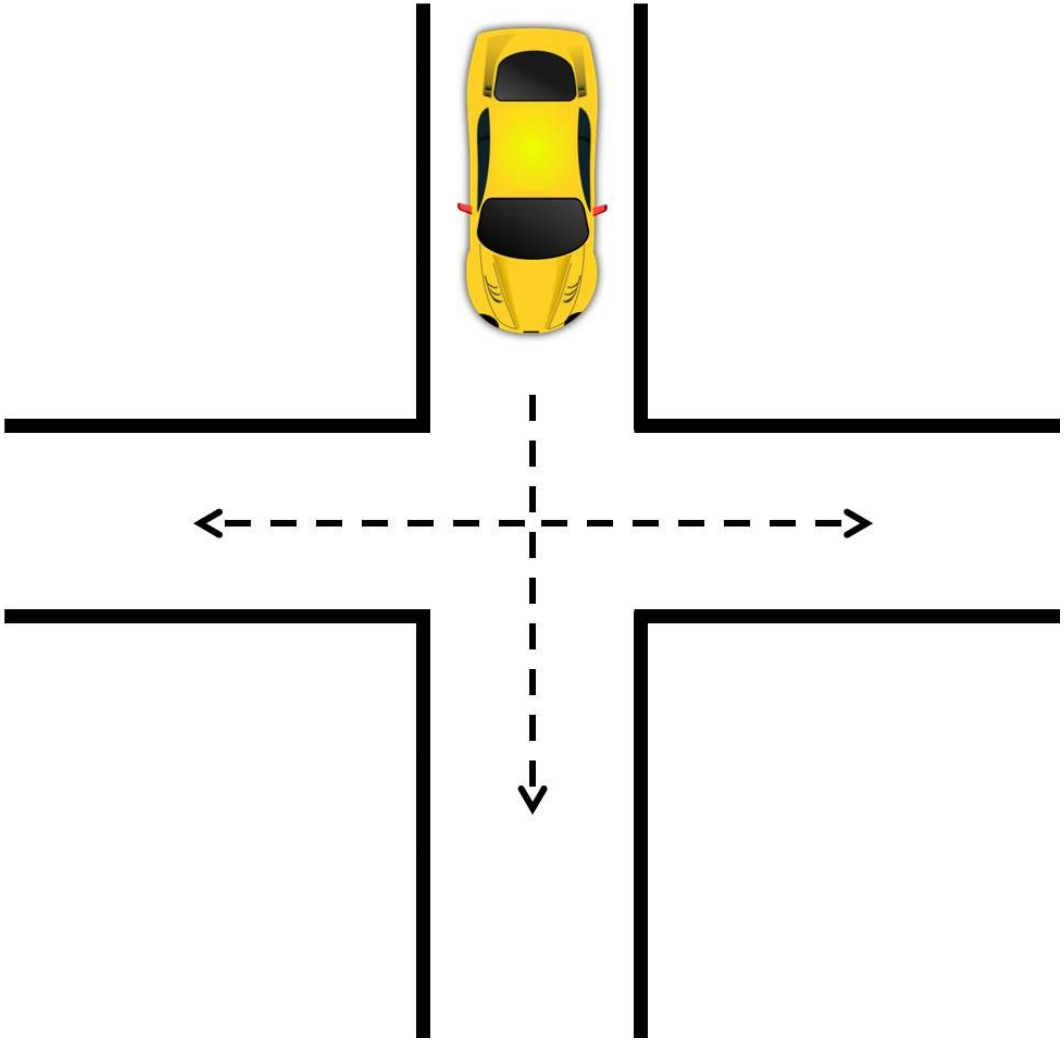

```
if average >= 70:
    print("You passed!")
    print("Get ready for summer vacation!")
else:
    print("You failed.")
    print("Get ready for summer school.")
```

Section 7.7

Multi-Way Selection

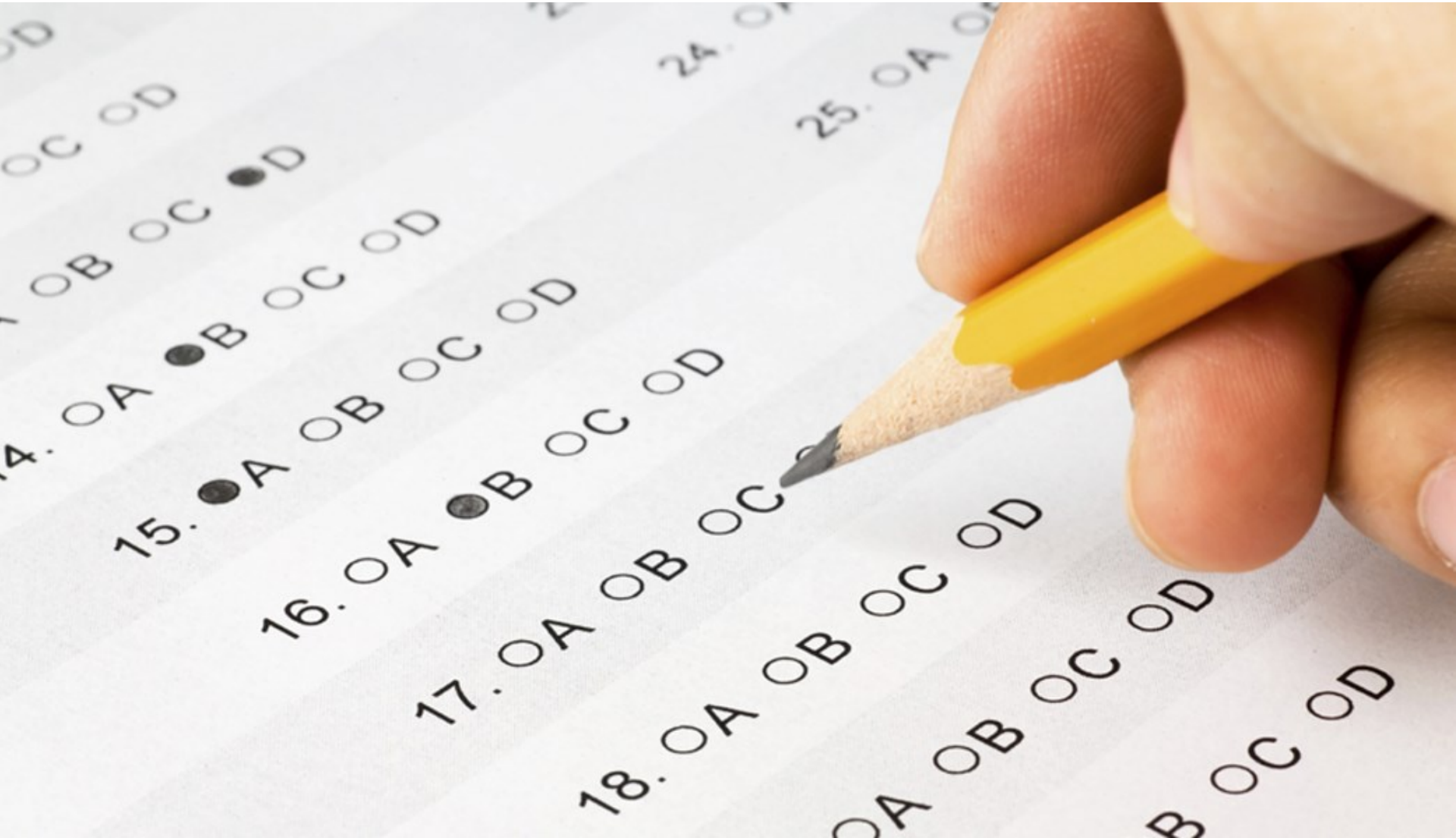
Multi-Way Selection

Real Life Example #1



Multi-Way Selection

Real Life Example #2



Multi-Way Selection

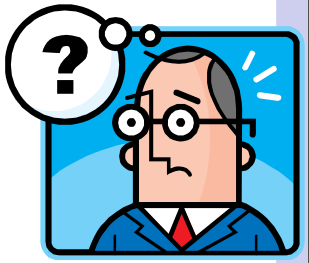
Real Life Example #3

```python
# Selection06.py
# This program is supposed to display the letter grade
# earned based on the number grade entered by the user.
# Since there are more than 2 possible paths (A,B,C,D,F)
# this would be an example of "Multi-Way Selection"...
# if the program worked; however, using 5 separate <if>
# statements has created a Logic Error with strange output.


print()
grade = eval(input("Enter Number Grade  -->  "))
print()

if grade >= 90:
   print("You earned an A!")
if grade >= 80:
   print("You earned a B.")
if grade >= 70:
   print("You earned a C.")
if grade >= 60:
   print("You earned a D.")
if grade >= 0:
   print("You earned an F.")
```

```python
# Selection06.py
# This program is supposed to d
# earned based on the number gr
# Since there are more than 2 p
# this would be an example of "
# if the program worked; howeve
# statements has created a Logi


print()
grade = eval(input("Enter Number Grade  -->  "))
print()

if grade >= 90:
    print("You earned an A!")
if grade >= 80:
    print("You earned a B.")
if grade >= 70:
    print("You earned a C.")
if grade >= 60:
    print("You earned a D.")
if grade >= 0:
    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
```



```python
# Selection07.py
# This program fixes the Logic Error of the
# previous program by adding several strategic
# <else> statements which will ensure that only
# 1 letter grade is displayed.  While this works,
# the program's indentation is somewhat annoying.

print()
grade = eval(input("Enter Number Grade  -->  "))
print()

if grade >= 90:
   print("You earned an A!")
else:
   if grade >= 80:
      print("You earned a B.")
   else:
      if grade >= 70:
         print("You earned a C.")
      else:
         if grade >= 60:
            print("You earned a D.")
         else:
            print("You earned an F.")
```

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

Enter Number Grade  -->  100

You earned an A!

----jGRASP: operation compl
```

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

Enter Number Grade  -->  65

You earned a D.

----jGRASP: operation compl
```

```python
grade = eval(input("Enter Number Grade  -->  "))
print()

if grade >= 90:
   print("You earned an A!")
else:
   if grade >= 80:
      print("You earned a B.")
   else:
      if grade >= 70:
         print("You earned a C.")
      else:
         if grade >= 60:
            print("You earned a D.")
         else:
            print("You earned an F.")
```

```
# Selection08.py
# This program shows a better way to do "Multi-Way
# Selection" using <if..elif..else>.  The <elif>
# command essentially combines the <else> with the
# next <if>.  Not only is this less code to type,
# it also has nicer indentation.


print()
grade = eval(input("Enter Number Grade  -->  "))
print()

if grade >= 90:
   print("You earned an A!")
elif grade >= 80:
   print("You earned a B.")
elif grade >= 70:
   print("You earned a C.")
elif grade >= 60:
   print("You earned a D.")
else:
   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
```

```python
print()
grade = eval(input("Enter Number Grade  -->  "))
print()

if grade >= 90:
    print("You earned an A!")
elif grade >= 80:
    print("You earned a B.")
elif grade >= 70:
    print("You earned a C.")
elif grade >= 60:
    print("You earned a D.")
else:
    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 print()
13 grade = input("Enter Letter Grade  -->  ")
14 print()
15
16 if grade == 'A':
17    print("You grade is 90 or above.")
18    print("Excellent!")
19 elif grade == 'B':
20    print("You grade is in the 80s.")
21    print("Good")
22 elif grade == 'C':
23    print("You grade is in the 70s.")
24    print("Fair")
25 elif grade == 'D':
26    print("You grade is in the 60s.")
27    print("Poor")
28 elif grade == 'F':
29    print("You grade is below 60.")
30    print("Bad")
```

```python
# Selection09.py
# This program demonstrates a numb
# 1. Selection can be based on tex
#    not just number values.
# 2. As with other selection struc
#    Selection can control multipl
#    commands as long as proper, c
#    indentation is used.
# 3. The program will not work pro
#    user does not enter an A, B,

print()
grade = input("Enter Letter Grade
print()

if grade == 'A':
    print("You grade is 90 or above
    print("Excellent!")
elif grade == 'B':
    print("You grade is in the 80s.
    print("Good")
elif grade == 'C':
    print("You grade is in the 70s.
    print("Fair")
elif grade == 'D':
    print("You grade is in the 60s.
    print("Poor")
elif grade == 'F':
    print("You grade is below 60.")
    print("Bad")
```

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

Enter Letter Grade  -->  A

You grade is 90 or above.
Excellent!

----jGRASP: operation com
```

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

Enter Letter Grade  -->  B

You grade is in the 80s.
Good

----jGRASP: operation com
```

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

Enter Letter Grade  -->  Q


----jGRASP: operation com
```

```
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  print()
8  grade = input("Enter Letter Grade  -->  ")
9  print()
10
11 if grade == 'A':
12    print("You grade is 90 or above.")
13    print("Excellent!")
14 elif grade == 'B':
15    print("You grade is in the 80s.")
16    print("Good")
17 elif grade == 'C':
18    print("You grade is in the 70s.")
19    print("Fair")
20 elif grade == 'D':
21    print("You grade is in the 60s.")
22    print("Poor")
23 elif grade == 'F':
24    print("You grade is below 60.")
25    print("Bad")
26 else:
27    print("You did not enter an A, B, C, D or F.")
28    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.
```

```
15    print("You grade is in the 80s.")
16    print("Good")
17 elif grade == 'C':
18    print("You grade is in the 70s.")
19    print("Fair")
20 elif grade == 'D':
21    print("You grade is in the 60s.")
22    print("Poor")
23 elif grade == 'F':
24    print("You grade is below 60.")
25    print("Bad")
26 else:
27    print("You did not enter an A, B, C, D or F.")
28    print("Please re-run the program and try again.")
```

Multi-Way Selection

General Syntax:

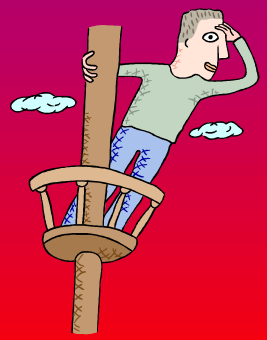
if *first condition is True*:
    execute first set of program statements
elif *second condition is True*:
    execute second set of program statements
: : : : :
else: *# when all above conditions are False*
    execute default set of program statements

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



```
# Selection11.py
# This program has 2 separate <if..else> structures.
# The first determines if a student is admitted to
# the college based on his/her SAT score.
# The second determines is that student qualifies
# for financial aid based on his/her family income.
# The problem with this program is that even if the
# student is not admitted, it still asks about family
# income and has the potential of telling a student who
# was not admitted that he/she qualifies for financial aid.

print()
sat = eval(input("Enter SAT score --> "))
print()

if sat >= 1100:
   print("You are admitted.")
   print("Orientation will start in June.")
else:
   print("You are not admitted.")
   print("Please try again when your SAT improves.")

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.")
```

```python
# Selection11.py
# This program has
# The first determ
# the college base
# The second deter
# for financial ai
# The problem with
# student is not a
# income and has t
# was not admitted

print()
sat = eval(input("
print()

if sat >= 1100:
    print("You are
    print("Orientat
else:
    print("You are
    print("Please t

print()
income = eval(inpu
print()

if income < 20000:
    print("You qual
else:
    print("You do n
```

```
----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.
```

```python
# Selection12.py
# This program fixes the issue of the previous program
# by "nesting" the second <if..else> structure inside
# the <if> part of the first.  Now, the "family income"
# question is only asked to students who are admitted.
# NOTE: Proper indentation is VERY important here.


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.")
```

```
----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
```

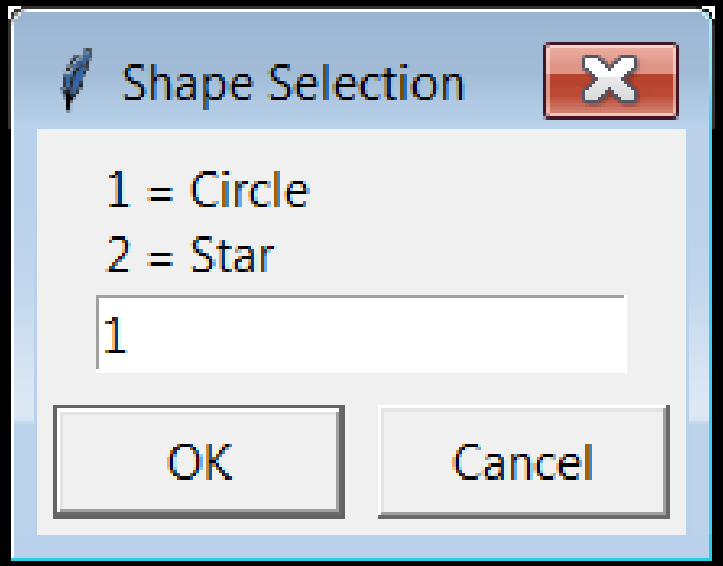
```
----jGRASP exec: python Se

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.
```

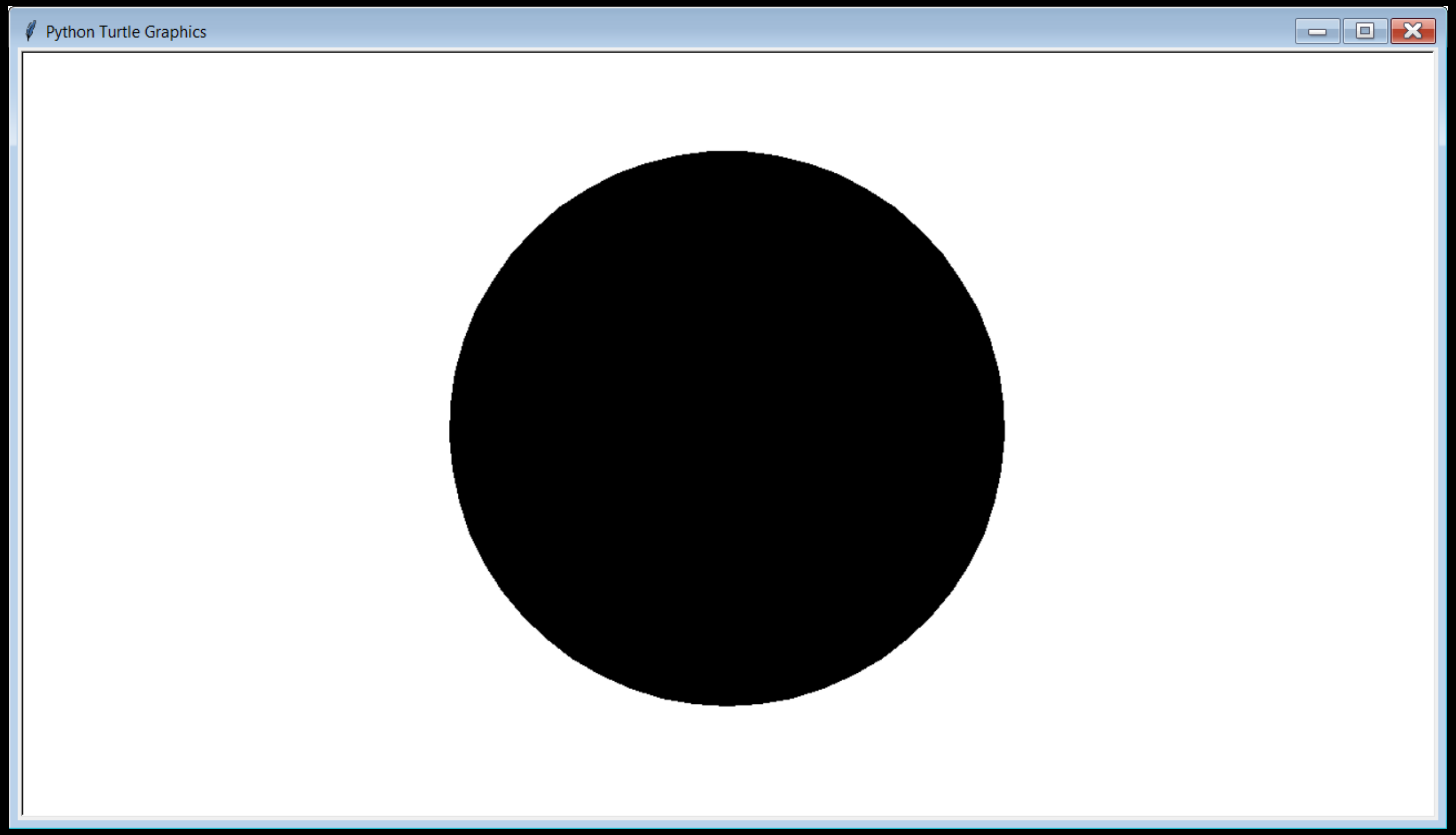
```
----jGRASP exec: python Selection12.py

Enter SAT score --> 700

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

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

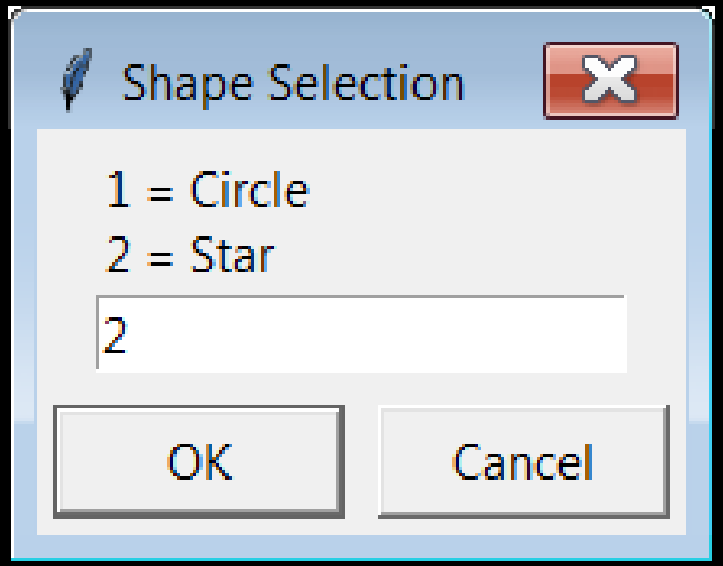
Section 7.9

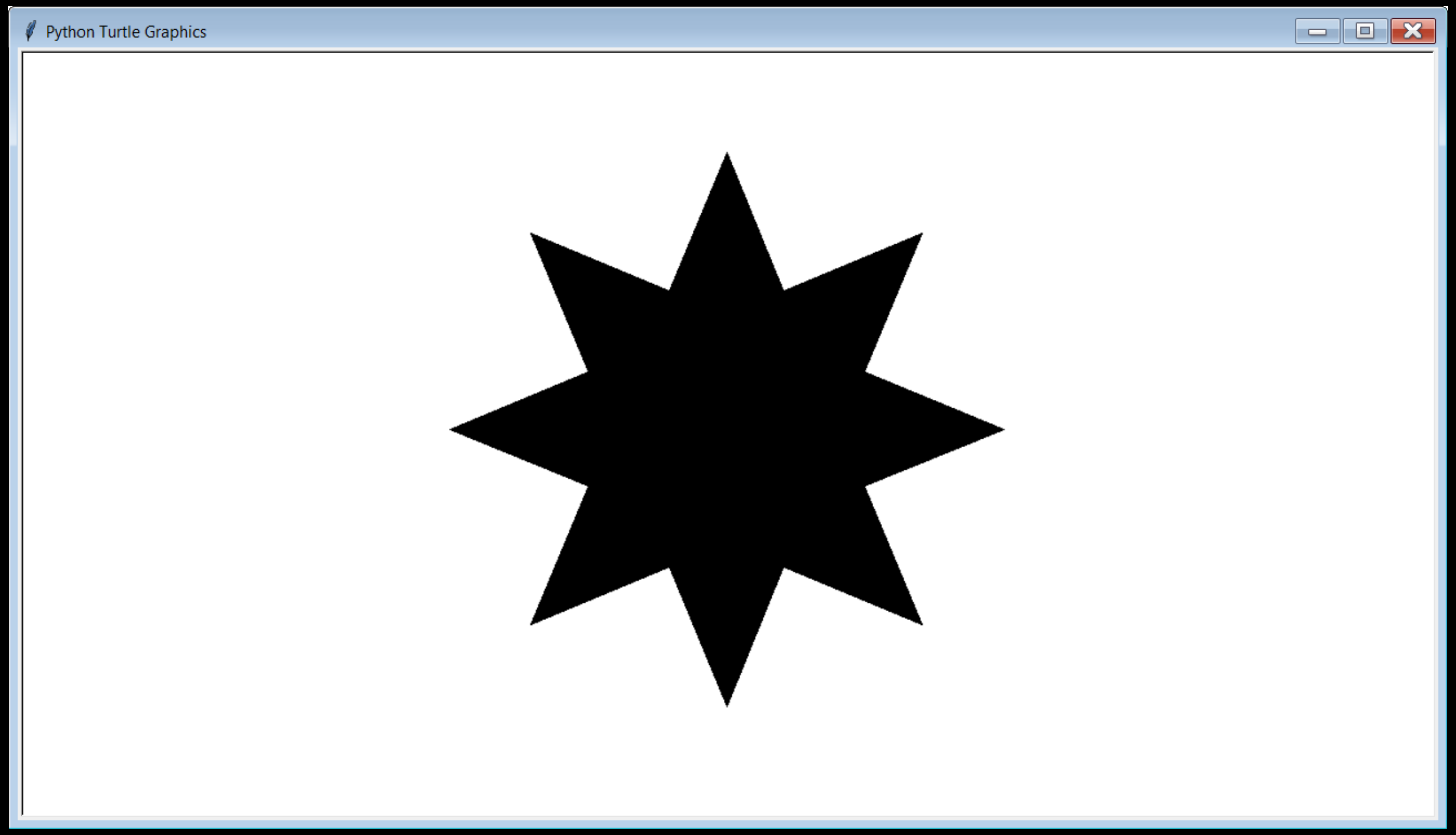
Combining Selection with Graphics

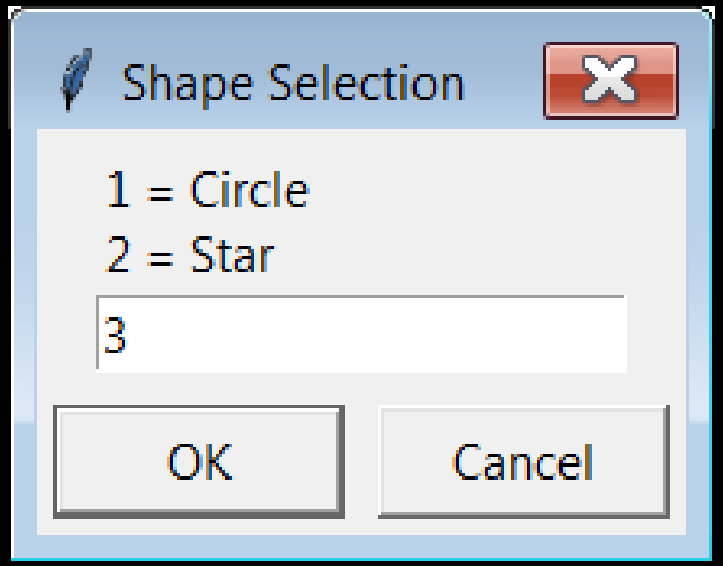
```
1 # Selection13.py
2 # This program demonstrates that selection can be used
3 # to manipulate the output of a graphics program.
4 # This is the very thing you will be doing in Lab 7B.
5
6
7 from Graphics import *
8
9 beginGrfx(1300,700)
10
11 shapeNum = numinput("Shape Selection","1 = Circle \n2 = 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()
```

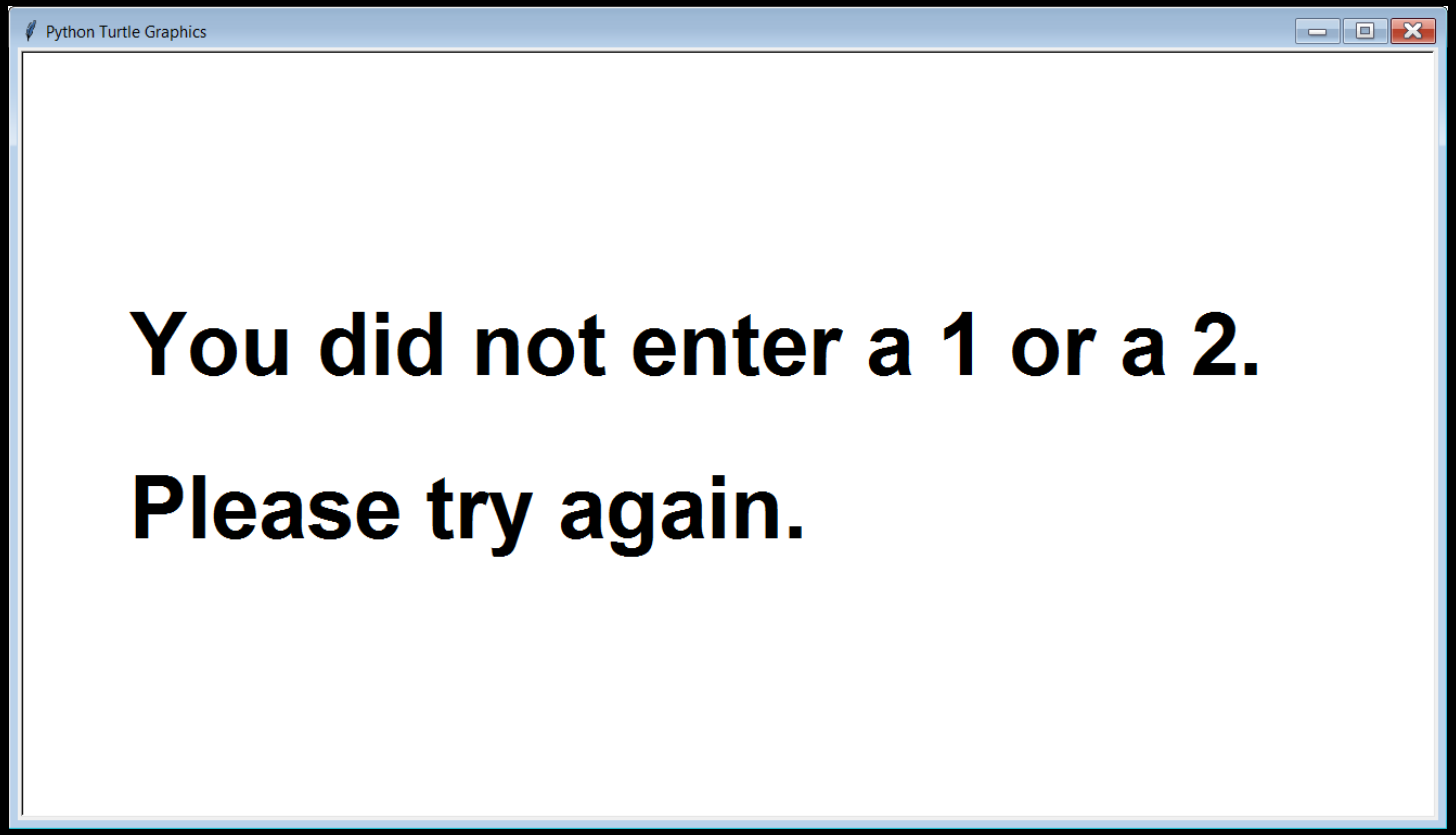

Shape Selection
1 = Circle
2 = Star
1
OK
Cancel

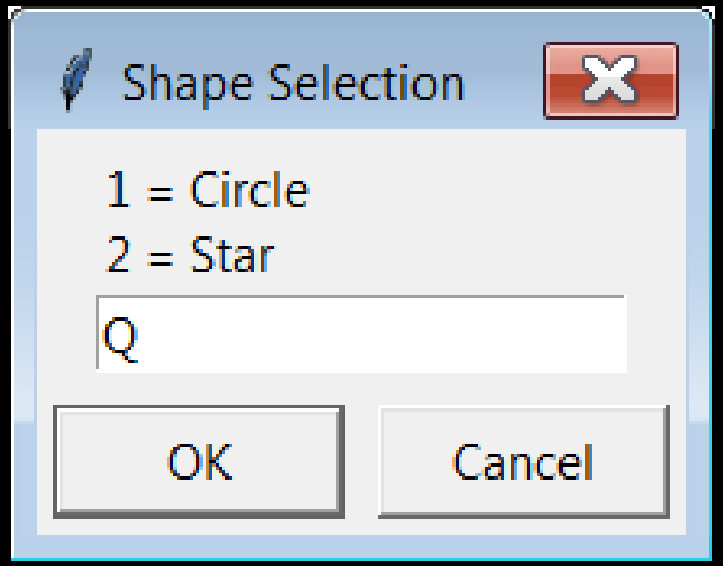

Python Turtle Graphics


Shape Selection
1 = Circle
2 = Star
2
OK
Cancel


Python Turtle Graphics


Shape Selection
1 = Circle
2 = Star
3
OK
Cancel


Python Turtle Graphics
You did not enter a 1 or a 2.
Please try again.


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

```python
# NumberFormat01.py
# This program demonstrates that by default
# numbers are displayed "left-justified"
# which means they do not line up by their
# place value.


print()
print(1)
print(12)
print(123)
print(1234)
print(12345)

```

```
----jGRASP

1
12
123
1234
12345

----jGRASP:
```

```python
# NumberFormat02.py
# This program demonstrates one way to properly line
# numbers up by place value using the <format> command.
# The 05 inside "{:05}" means each number will have
# enough 0s placed at the front of the number to force
# it to be displayed as a 5 digit number.


print()
print("{:05}".format(1))
print("{:05}".format(12))
print("{:05}".format(123))
print("{:05}".format(1234))
print("{:05}".format(12345))

```

```
----jGRASP

00001
00012
00123
01234
12345

----jGRASP:
```

```python
# NumberFormat03.py
# This program demonstrates that leading 0s are
# not required to line up numbers by place value.
# Just leave out the 0, and the number will be
# displayed with leading spaces instead.


print()
print("{:5}".format(1))
print("{:5}".format(12))
print("{:5}".format(123))
print("{:5}".format(1234))
print("{:5}".format(12345))

```

```
----jGRASP

    1
   12
  123
 1234
12345

----jGRASP:
```

```python
# NumberFormat04.py
# This program demonstrates what happens when the
# format size for the number is not large enough.
# If the format is not possible, it is simply
# ignored.  There is no error message.


print()
print("{:5}".format(1))
print("{:5}".format(12))
print("{:5}".format(123))
print("{:5}".format(1234))
print("{:5}".format(12345))
print("{:5}".format(123456))
print("{:5}".format(1234567))
print("{:5}".format(12345678))
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 <u>appearance</u> of a number, it cannot change the <u>value</u> 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.

```python
# NumberFormat05.py
# This program demonstrates how to add commas (,)
# as a "thousand separator".  It may seem like
# it stops working at 10 million.  The problem is
# the commas count as digits.


print()
print("{:9,}".format(1))
print("{:9,}".format(12))
print("{:9,}".format(123))
print("{:9,}".format(1234))
print("{:9,}".format(12345))
print("{:9,}".format(123456))
print("{:9,}".format(1234567))
print("{:9,}".format(12345678))
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 <u>characters</u> which includes digits, commas, and even the decimal point.

```python
# NumberFormat06.py
# This program fixes the problem of the previous
# program by increasing the total character count
# to accommodate the commas.


print()
print("{:11,}".format(1))
print("{:11,}".format(12))
print("{:11,}".format(123))
print("{:11,}".format(1234))
print("{:11,}".format(12345))
print("{:11,}".format(123456))
print("{:11,}".format(1234567))
print("{:11,}".format(12345678))
print("{:11,}".format(123456789))
```

```
----jGRASP

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

----jGRASP
```

```python
# NumberFormat07.py
# This program demonstrates that commas can
# be used without leading zeros or spaces.


print()
print("{:,}".format(1))
print("{:,}".format(12))
print("{:,}".format(123))
print("{:,}".format(1234))
print("{:,}".format(12345))
print("{:,}".format(123456))
print("{:,}".format(1234567))
print("{:,}".format(12345678))
print("{:,}".format(123456789))
```

```
----jGRASP

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

----jGRASP
```

```python
# NumberFormat08.py
# This program repeats Documentation02.py from Chapter 4.
# One problem with the program still has is that the
# numbers which are displayed represent money, but they
# are not rounded to 2 decimal places.


hoursWorked = 35
hourlyRate = 8.75
grossPay = hoursWorked * hourlyRate
deductions = grossPay * 0.29
netPay = grossPay - deductions

print()
print("Hours Worked: ",hoursWorked)
print("Hourly Rate:  ",hourlyRate)
print("Gross Pay:    ",grossPay)
print("Deductions:   ",deductions)
print("Net Pay:      ",netPay)
```

```python
# NumberFormat08.py
# This program repeats Do
# One problem with the pr
# numbers which are displ
# are not rounded to 2 de


hoursWorked = 35
hourlyRate = 8.75
grossPay = hoursWorked * hourlyRate
deductions = grossPay * 0.29
netPay = grossPay - deductions

print()
print("Hours Worked: ",hoursWorked)
print("Hourly Rate:  ",hourlyRate)
print("Gross Pay:    ",grossPay)
print("Deductions:   ",deductions)
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
```

```
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))
```

```python
# NumberFormat09.py
# This program demonstrat
# output with the <format
# "{:6.2f}" means this is
# which is the same thing
# The 6 indicates the ent
# long, including the dec
# there will be 2 digits


hoursWorked = 35
hourlyRate = 8.75
grossPay = hoursWorked * hourlyRate
deductions = grossPay * 0.29
netPay = grossPay - deductions

print()
print("Hours Worked:","{:6.2f}".format(hoursWorked))
print("Hourly Rate: ","{:6.2f}".format(hourlyRate))
print("Gross Pay:   ","{:6.2f}".format(grossPay))
print("Deductions:  ","{:6.2f}".format(deductions))
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
```

```python
# NumberFormat10.py
# This program demonstrates how to format
# real numbers without leading spaces.


hoursWorked = 35
hourlyRate = 8.75
grossPay = hoursWorked * hourlyRate
deductions = grossPay * 0.29
netPay = grossPay - deductions

print()
print("Hours Worked:","{:.2f}".format(hoursWorked))
print("Hourly Rate: ","{:.2f}".format(hourlyRate))
print("Gross Pay:   ","{:.2f}".format(grossPay))
print("Deductions:  ","{:.2f}".format(deductions))
print("Net Pay:     ","{:.2f}".format(netPay))

```

```python
# NumberFormat10.py
# This program demonstr
# real numbers without


hoursWorked = 35
hourlyRate = 8.75
grossPay = hoursWorked
deductions = grossPay * 0.29
netPay = grossPay - deductions

print()
print("Hours Worked:","{:.2f}".format(hoursWorked))
print("Hourly Rate: ","{:.2f}".format(hourlyRate))
print("Gross Pay:   ","{:.2f}".format(grossPay))
print("Deductions:  ","{:.2f}".format(deductions))
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
```

```
# NumberFormat11.py
# This program demonstrates that real numbers can be
# formatted with leading spaces, rounded digits past
# the decimal point, commas, and even a dollar $ign.

hoursWorked = 50
hourlyRate = 199.98
grossPay = hoursWorked * hourlyRate
deductions = grossPay * 0.29
netPay = grossPay - deductions

print()
print("Hours Worked:",hoursWorked)
print("Hourly Rate: ","${:8,.2f}".format(hourlyRate))
print("Gross Pay:   ","${:8,.2f}".format(grossPay))
print("Deductions:  ","${:8,.2f}".format(deductions))
print("Net Pay:     ","${:8,.2f}".format(netPay))

```

```python
# NumberFormat11.py
# This program demonstr
# formatted with leadi
# the decimal point, c

hoursWorked = 50
hourlyRate = 199.98
grossPay = hoursWorked
deductions = grossPay * 0.29
netPay = grossPay - deductions

print()
print("Hours Worked:",hoursWorked)
print("Hourly Rate: ","${:8,.2f}".format(hourlyRate))
print("Gross Pay:   ","${:8,.2f}".format(grossPay))
print("Deductions:  ","${:8,.2f}".format(deductions))
print("Net Pay:     ","${: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
```