Lecture 06: Decision Structure

Rawls Profess of MIS
Jaeki Song, Ph.D

Topics

The main topics:

- The `if` Statement
- The `if-else` Statement
- Nested `if` statements
- The `if-else-if` Statement
- Logical Operators

3-2

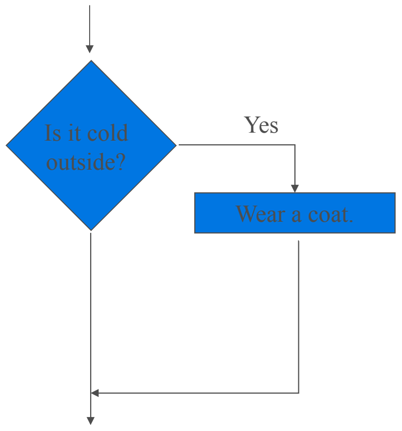
The `if` Statement

- Sequence structure
- Decision structure
  - specific action(s) performed only if a condition exists
- The ***`if`*** statement decides whether a section of code executes or not.
  - Uses a `boolean` to decide whether the next statement or block of statements executes.
  - Python syntax

```
if boolean expression is true:
  execute next statement
```

Flowcharts

- If statements can be modeled as a flow chart.

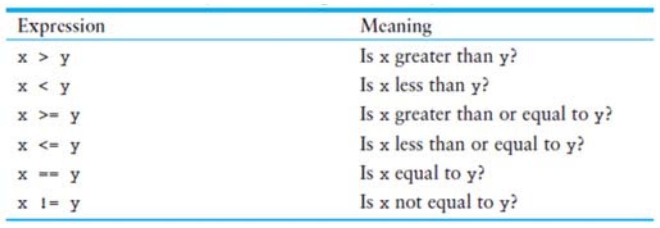


Boolean Expression

- expression tested by if statement to determine if it is true or false
- Relational operators
  - determines whether a specific relationship exists between two values
  - Example: a > b
    - `true` if a is greater than b; `false` otherwise

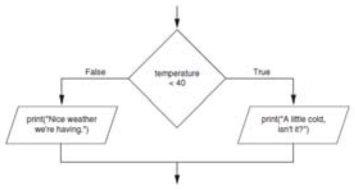
Boolean Expression

- Relational operators



Expression	Meaning
`x > y`	Is x greater than y?
`x < y`	Is x less than y?
`x >= y`	Is x greater than or equal to y?
`x <= y`	Is x less than or equal to y?
`x == y`	Is x equal to y?
`x != y`	Is x not equal to y?

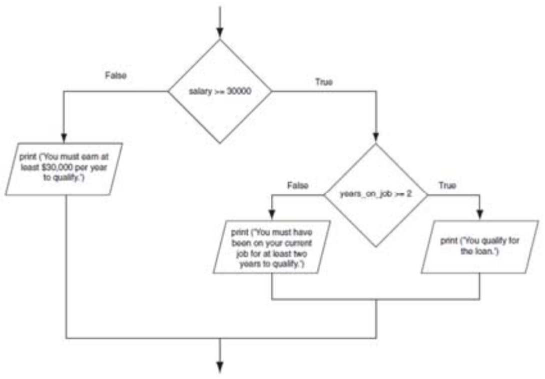
if-else Statement

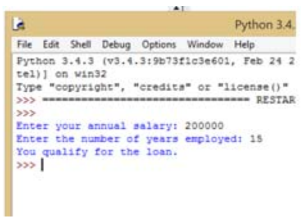

temperature < 40
False
True
print("Nice weather we're having.")
print("A little cold, isn't it?")

- Dual alternative decision structure
  - two possible paths of execution
  - One is taken if the condition is true, and the other if the condition is false
  - Syntax:

```
if condition:
    statements
else:
    other statements
```

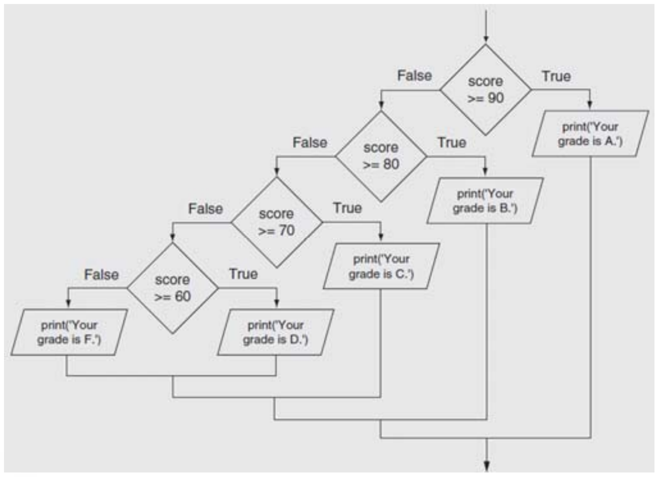
Nested Decision Structure


salary >= 30000
False
True
print ("You must earn at least $30,000 per year to qualify.")
years_on_job >= 2
False
True
print ("You must have been on your current job for at least two years to qualify.")
print ("You qualify for the loan.")

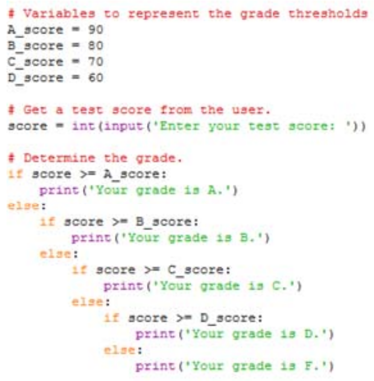


```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2
tel)] on win32
Type "copyright", "credits" or "license()"
>>> ================================ RESTAR
>>>
Enter your annual salary: 200000
Enter the number of years employed: 15
You qualify for the loan.
>>>
```

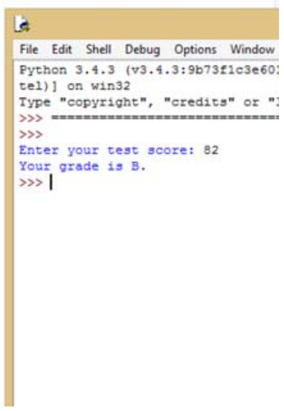
Testing a Series of Conditions


score >= 90
True
False
print('Your grade is A.')
score >= 80
True
False
print('Your grade is B.')
score >= 70
True
False
print('Your grade is C.')
score >= 60
True
False
print('Your grade is D.')
print('Your grade is F.')

Testing a Series of Conditions



```
# Variables to represent the grade thresholds
A_score = 90
B_score = 80
C_score = 70
D_score = 60

# Get a test score from the user.
score = int(input('Enter your test score: '))

# Determine the grade.
if score >= A_score:
    print('Your grade is A.')
else:
    if score >= B_score:
        print('Your grade is B.')
    else:
        if score >= C_score:
            print('Your grade is C.')
        else:
            if score >= D_score:
                print('Your grade is D.')
            else:
                print('Your grade is F.')
```



```
Python 3.4.3 (v3.4.3:9b73f1c3e60
tel)] on win32
Type "copyright", "credits" or "
>>> ================================
>>>
Enter your test score: 82
Your grade is B.
>>> 
```

Testing a Series of Conditions

- If-elif-else statement
  - A special version of the decision structure
  - Example:

```
if score >= A_score:
    print('Your grade is A.')
elif score >= B_score:
    print('Your grade is B.')
elif score >= C_score
    print('Your grade is C.')
elif score >= D_score:
    print('Your grade is D.')
else:
    print('Your grade is F.')
```

Logical Operators

- operators that can be used to create complex Boolean expressions
  - `and` operator and `or` operator: binary operators, connect two Boolean expressions into a compound Boolean expression
  - `not` operator: unary operator, reverses the truth of its Boolean operand

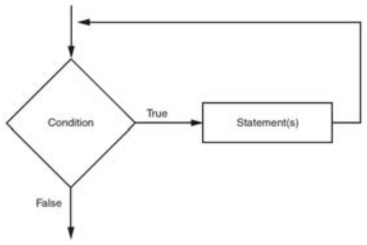
Assignment

- Blackboard
  - In-class 6-1

Repetition

- Often have to write code that performs the same task multiple times
  - Disadvantages to duplicating code
    - Makes program large
    - Time consuming
    - May need to be corrected in many places
- Repetition structure: makes computer repeat included code as necessary
  - Includes condition-controlled loops and count-controlled loops

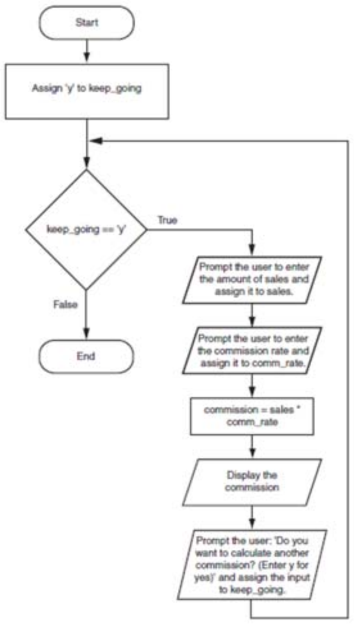
Condition-Controlled Loop

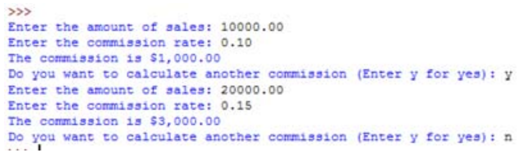


- `while` loop
  - while condition is true, do something
    - Two parts:
      - Condition tested for true or false value
      - Statements repeated as long as condition is true
    - In flow chart, line goes back to previous part
    - General format:

```
while condition:
    statements
```

Practice: Commission


Start
Assign 'y' to keep_going
keep_going == 'y'
True
False
End
Prompt the user to enter the amount of sales and assign it to sales.
Prompt the user to enter the commission rate and assign it to comm_rate.
commission = sales * comm_rate
Display the commission
Prompt the user: 'Do you want to calculate another commission? (Enter y for yes)' and assign the input to keep_going.



```
>>>
Enter the amount of sales: 10000.00
Enter the commission rate: 0.10
The commission is $1,000.00
Do you want to calculate another commission (Enter y for yes): y
Enter the amount of sales: 20000.00
Enter the commission rate: 0.15
The commission is $3,000.00
Do you want to calculate another commission (Enter y for yes): n
...
```

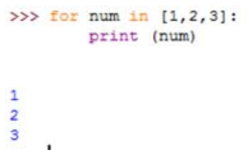
Infinite Loop

- Loops must contain within themselves a way to terminate
  - Something inside a `while` loop must eventually make the condition false
- *Infinite loop*
  - loop that does not have a way of stopping
    - Repeats until program is interrupted
    - Occurs when programmer forgets to include stopping code in the loop

Count-Controlled Loop

- *for* loop
  - iterates a specific number of times
  - Use a for statement to write count-controlled loop
    - Designed to work with sequence of data items
      - Iterates once for each item in the sequence
    - General format:

```
for variable in [val1, val2, etc]:
        statements
```



```
>>> for num in [1,2,3]:
        print (num)

1
2
3
```

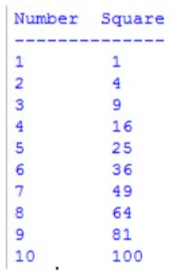
- Target variable: the variable which is the target of the assignment at the beginning of each iteration

Using the `range` Function with the ***for*** Loop

- The ***range*** function simplifies the process of writing a `for` loop
  - `range` returns an iterable object
    - iterable: contains a sequence of values that can be iterated over
- `range` characteristics:
  - One argument: used as ending limit
  - Two arguments: starting value and ending limit
  - Three arguments: third argument is step value

Using the Target Variable Inside the Loop

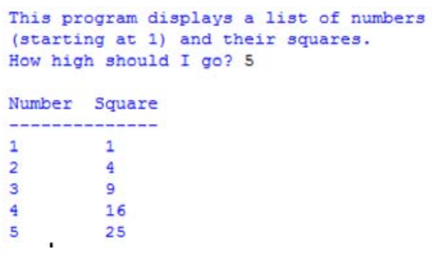
- Purpose of target variable is to reference each item in a sequence as the loop iterates
- Target variable can be used in calculations or tasks in the body of the loop
  - Example:



```
Number   Square
--------------
1         1
2         4
3         9
4         16
5         25
6         36
7         49
8         64
9         81
10        100
```

Letting the User Control the Loop Iterations

- Sometimes the programmer does not know exactly how many times the loop will execute
- Can receive range inputs from the user, place them in variables, and call the `range` function in the for clause using these variables
  - Be sure to consider the end cases: `range` does not include the ending limit



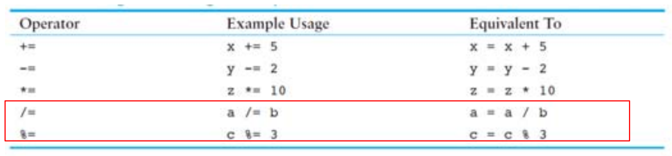
```
This program displays a list of numbers
(starting at 1) and their squares.
How high should I go? 5

Number   Square
--------------
1         1
2         4
3         9
4         16
5         25
```

Assignment

- Blackboard
  - InClass 6-2

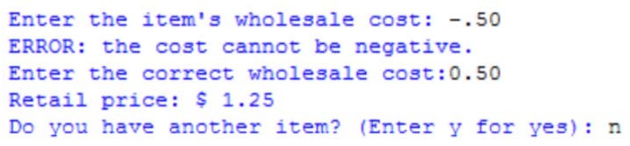
Operators

- The augmented assignment operators



Operator	Example Usage	Equivalent To
+=	x += 5	x = x + 5
-=	y -= 2	y = y - 2
*=	z *= 10	z = z * 10
/=	a /= b	a = a / b
%=	c %= 3	c = c % 3

Input validation



```
Enter the item's wholesale cost: -.50
ERROR: the cost cannot be negative.
Enter the correct wholesale cost:0.50
Retail price: $ 1.25
Do you have another item? (Enter y for yes): n
```