

Introduction to Programming

Python Lab 7: if Statement

Getting Started

- Create a new folder in your disk space with the name **PythonLab7**
- Launch the Python Integrated Development Environment (IDLE) - begin with the **Start** icon in the lower left corner of the screen.
- If you are in a DCSIS laboratory, search using the keyword **Python** and click on **IDLE (Python 3.6 64-bit)**

A window with the title **Python 3.6.2** should appear. This window is the *Shell*.

Getting Started (2)

- If you are in the ITS laboratory MAL 109, then right mouse click on the **Start** icon in the lower left corner of the screen.

A list of menu options should appear and click on **Search**. Type **Python** in the search text box at the bottom of the pop-up window. A list of Apps should appear and select

Python 3.4 IDLE(PythonGUI)

A window with the title **Python 3.4.3 Shell** should appear. This window is the **Shell**.

- In the **Shell** click on **File**. A drop down menu will appear. Click on **New File**. A window with the `title` **Untitled** should appear. This window is the **Editor**.

Getting Started (3)

- In the *Editor*, click on **File**, and then in the drop down menu click on **Save As...** .

A window showing a list of folders should appear.

- To search any folder on the list, double click on the folder.
- Find the folder **PythonLab7** and double click on it.
- In the box **File name** at the bottom of the window
 1. Type `QuizGrading.py`
 2. Then click on the button **Save** in the lower right corner of the window.

The title of the *Editor* should change to show the location of the file `QuizGrading.py`.

Objectives of the exercises set

- Understand the use of multiple `if` statements to solve problems that have several levels of decision making.

Python provides the special construct `elif` for creating `if` statements containing multiple branches (selections).

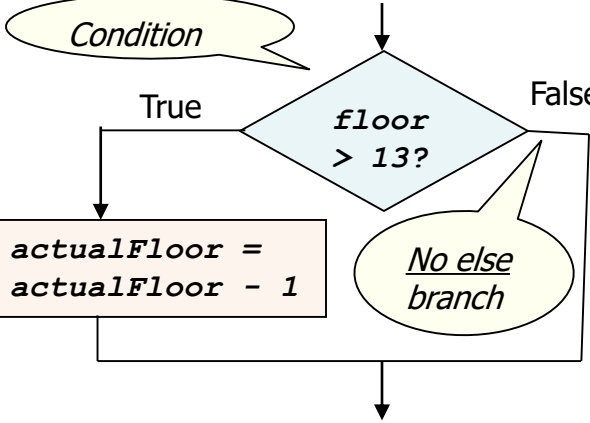
When using multiple `if` statements, we usually test general conditions after we test the more specific conditions first.

- Use relational operators in conditions that involve comparing two values.

| Python relational operators | Description |
|-----------------------------|-----------------------|
| < | Less than |
| <= | Less than or equal |
| > | Greater than |
| >= | Greater than or equal |
| == | Equal |
| != | Not equal |

Objectives of the exercises set (2)

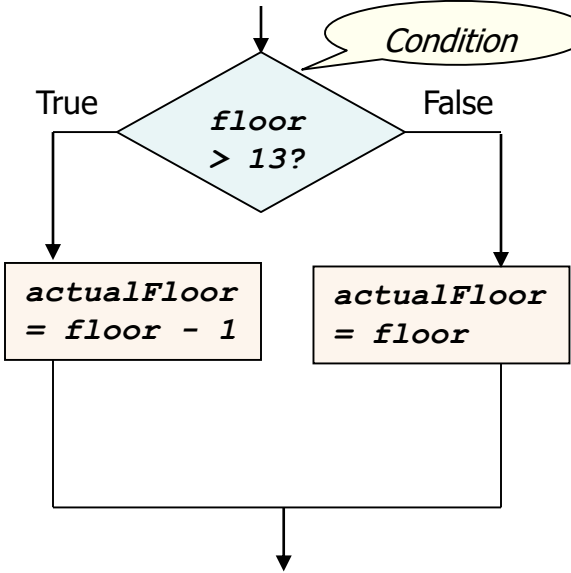
- An `if` statement is used to implement a decision. When a condition is satisfied (that is, `True`), one set of statements is executed. Otherwise, another set of statements is executed.
- The syntax of an `if` statement with No else branch is shown below. The **colon** indicates the header.

| Syntax | Example | Flow chart for <code>if</code> statement with No else branch |
|--|---|---|
| <p><code>if</code> condition :</p> <p>statement(s)</p> | <pre> floor = int(input("Floor: ")) actualFloor = floor if floor > 13 : actualFloor = actualFloor - 1 # True branch - execute the # statement above only if the # condition is True </pre> |  <pre> graph TD Start(()) --> Decision{floor > 13?} Decision -- True --> Process[actualFloor = actualFloor - 1] Decision -- False --> End(()) Process --> End </pre> |

 **Note:** Indent the block of statement(s) in the `True` branch.

Objectives of the exercises set (3)

- The syntax of an `if` statement with `else` branch is shown below. The **colon** indicates a header.

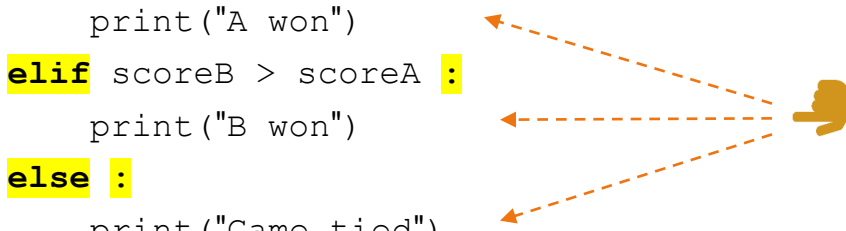
| Syntax | Example | Flow chart for <code>if</code> statement with <code>else</code> branch |
|--|--|---|
| <pre> if condition : statement(s) else : statement(s) </pre> | <pre> floor = int(input("Floor: ")) actualFloor = 0 if floor > 13 : actualFloor = floor - 1 # True branch - execute # only if the condition # is True else : actualFloor = floor # False branch - execute # only if the condition # is False </pre> |  <pre> graph TD Start(()) --> Decision{floor > 13?} Decision -- True --> TrueBox[actualFloor = floor - 1] Decision -- False --> FalseBox[actualFloor = floor] TrueBox --> End(()) FalseBox --> End </pre> |


Note: Align `if` and `else`

👉 Indent the block of `statement(s)` in each branch.

Objectives of the exercises set (4)

- The syntax of an `elif` statement is as follows.

| Syntax | Example |
|---|--|
| <pre> if condition : statement(s) elif condition : statement(s) else : statement(s) </pre> | <pre> scoreA = int(input("Enter a score for player A: ")) scoreB = int(input("Enter a score for player B: ")) if scoreA > scoreB : print("A won") elif scoreB > scoreA : print("B won") else : print("Game tied") </pre>  |

 **Note:** Align `if`, `elif` and `else`

Indent the statements in each branch.

- Understand the use of the remainder operator `%` in an arithmetic expression.

Program QuizGrading.py: Quiz grading

- **Question 2: Problem statement**

Write a program that inputs an integer from the keyboard. This integer is the score. Use an appropriate prompt.

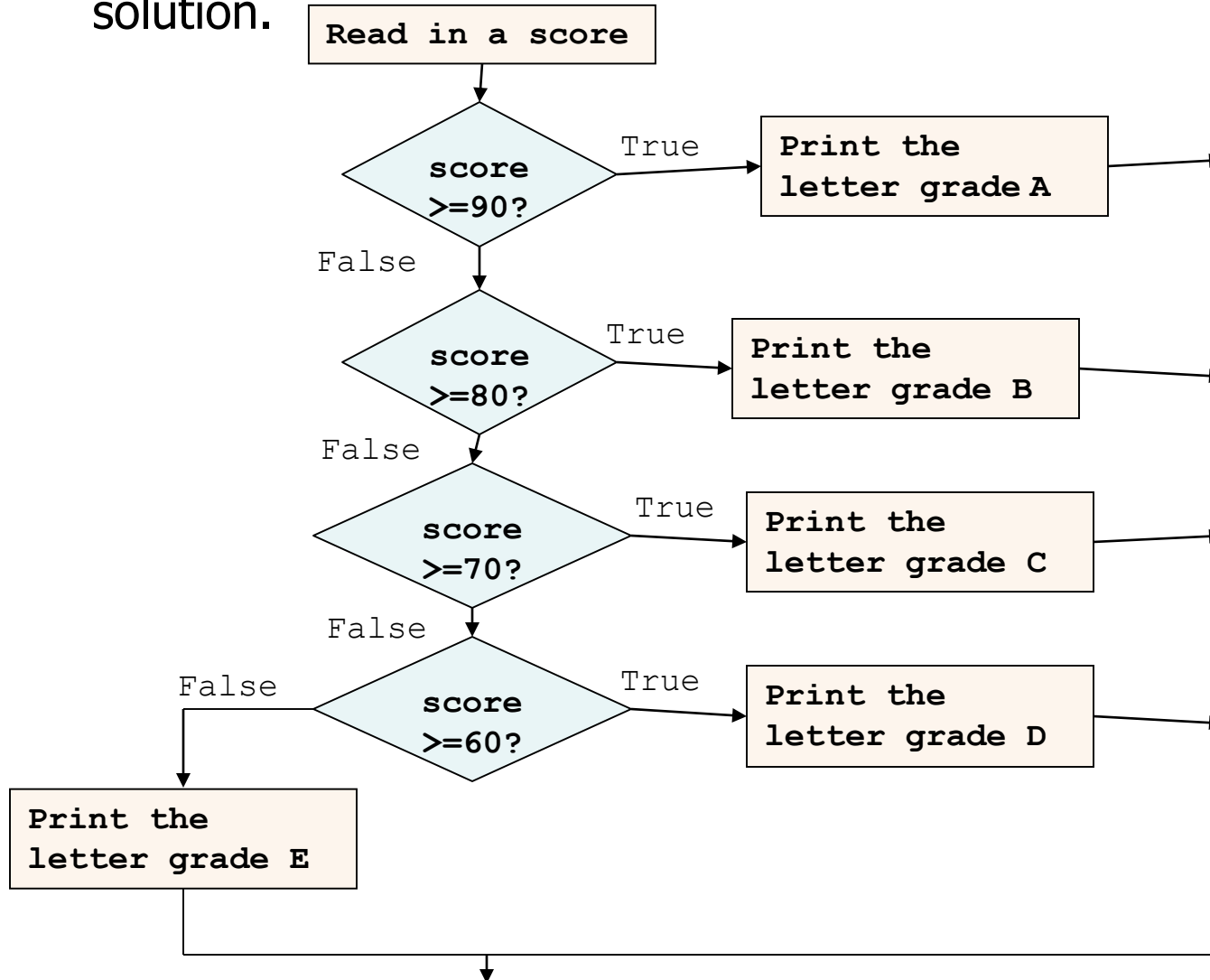
A letter grade is assigned to the score, according to the following table.

| Score | Grade |
|--------|-------|
| 90-100 | A |
| 80-89 | B |
| 70-79 | C |
| 60-69 | D |
| <60 | E |

Include in your program a statement to print the letter grade, together with an appropriate description. See Python for Everyone, R3.18.

Program QuizGrading.py: Quiz grading (2)

- **Problem solving** – the flow chart below illustrates a possible solution.



Program QuizGrading.py: Quiz grading (3)

- **Problem solving** - Convert the pseudo code below into a sequence of Python statements in your program.

Input

1. Read in an integer and store it in the variable `score`*

2. Write the statements below to check if the `score` is greater than or equal to 90, and then print the letter grade `A` in the True branch.

```
if score >= 90 :  
    print("Grade A")           # True branch
```



3. Add the following statements to check if the `score` is greater than or equal to 80, and then print the letter grade `B`.

```
elif score >= 80 :  
    print("Grade B")           # True branch
```



4. Write `elif` and `print` statements similar to **step 3** to check if the `score` is greater than or equal to 70, and then print the letter grade `C`.

Process the input and display the correct output (steps 2 to 6).

- ***Hint:** First use the `input` function to read in a numeric value typed in at the keyboard. Then use the function `int` to convert the input string to an integer and store it in the variable `score`.

Program QuizGrading.py: Quiz grading (4)

- **Problem solving** (continued)

5. Write `elif` and `print` statements similar to **step 3** to check if the `score` is greater than or equal to 60, and then print the letter grade `D`.
6. Lastly, add the statements below to print the letter grade `E` using the `else` statement.

```
else :  
    print("Grade E")
```



- Provide a comment at the beginning of the program to explain the purpose of the program together with your name and the date.
- Save the program to the file `QuizGrading.py` and then run it.

Note: Align `if`, `elif` and `else`. You must add a **colon** at the end of the statement. You must also indent the statement block in each branch so that it is part of the `if`, `elif` or `else` statement.

Program LeapYear.py: Leap year

- Create a new Editor for a new file called `LeapYear.py`

- **Question 3: Problem statement**

A year with 366 days is called a leap year. Usually years that are divisible by 4 are leap years, for example, 1996. However, years that are divisible by 100 are not leap years, unless the year is also divisible by 400.

Write a program that asks the user for a year and computes whether the year is a leap year. Use an appropriate print statement to display the result of the computation.

See Python for Everyone, P3.27.

Program LeapYear.py: Leap year (2)

- **Problem solving** - The following method can be used to check:
 - If a year is divisible by 4 but not by 100, it is a leap year.
 - If a year is divisible by 4 and by 100, it is not a leap year unless it is also divisible by 400.
- Three conditions to consider - we usually start with the tests for the more specific conditions first before testing the general conditions.

```
(year%400) == 0           # First condition tests if
                           # year is divisible by 400
(year%100) == 0          # Second condition tests if
                           # year is divisible by 100
(year%4) == 0            # Third condition tests if
                           # year is divisible by 4
```

Program LeapYear.py: Leap year (3)

- **Problem solving** - Convert the pseudo code below into a sequence of Python statements in your program.

Input

1. Read in an integer and store it in the variable `year`

Process the input and display the correct output (steps 2 to 5).

2. Write the statements below to check if the `year` is divisible by 400, and then print out the message, `Leap year`

```
if year%400 == 0 :  
    print("Leap year")           # True branch
```



3. Add the statements below to check if the `year` is divisible by 100, and then print the message, `Not a leap year`, in the `True` branch.

```
elif year%100 == 0 :  
    print("Not a leap year")     # True branch
```



4. Write `elif` and `print` statements similar to **step 3** to check if the `year` is divisible by 4, and then print the message, `Leap year`

Program LeapYear.py: Leap year (4)

- **Problem solving** (continued)

5. Lastly, add the statements below to print the message, Not a leap year

```
else :  
    print("Not a leap year")
```



- Provide a comment at the beginning of the program to explain the purpose of the program together with your name and the date.
- Save the program to the file `LeapYear.py` and then run it

Supplementary Questions for Private Study

- The laboratory worksheet contains supplementary questions in section 4 for private study.
- You are encouraged to complete the supplementary questions at home, or in the laboratory if you have time after completing questions 2 to 3.

Appendix A Testing different user inputs for the leap year problem

- The following table shows some user inputs for the year and the result of the computation.

| User input for the year | <code>year%400 == 0</code> | <code>year%100 == 0</code> | <code>year%4 == 0</code> | result |
|--------------------------------|-----------------------------------|-----------------------------------|---------------------------------|-----------------|
| 2016 | False | False | True | Leap year |
| 1600 | True | True | True | Leap year |
| 1800 | False | True | True | Not a leap year |