Python 2.7: Conditional Execution

Lecture notes of Alexander Wood
awood@citytech.cuny.edu

New York City College of Technology

Code Structures

- Many languages use curly braces or characters such as `begin` and `end` to designate sections of code
- Recall that Python uses **indentation to define a program's structure**
- You can use any indentation you want but Python expects you to be consistent. The recommended style is four spaces.

Code Structures: Comments

Write **comments**, pieces of text that the program interpreter ignores, using the hashtag.

```
# This is a comment
# Comments are SINGLE-LINE
```

However, you can write multiline quotes in your code and then not print them to write multiple lines of unread text.

```
'''
    You can write a multiline
    "comment" by using triple
    quotes, and not printing
    the quote.
'''
```

A hashtag inside of a print statement is NOT a comment.

```
>>> print('A hashtag # inside a print statement')
A hashtag # inside a print statement
```

Decision Structures

Code is executed sequentially, or line-by-line. We can control which lines of code are executed with **decision structures**.

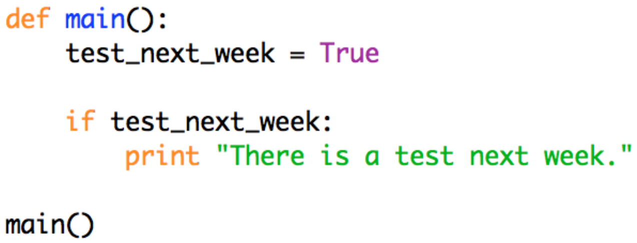
The if Statement

The `if` statement is the first example of a decision structure we shall see. It has the following syntax:

```
if condition:
    statement
    statement
    statement
```

The if statement

The `if` statement checks a condition is true. If the condition is true, the statements inside of the if condition are executed.



```
def main():
    test_next_week = True

    if test_next_week:
        print "There is a test next week."

main()
```

The pass statement

There is no limit on the number of statements that can be in an if statement, but there has to be at least one. You can use the **pass** statement as a placeholder for code you will fill in later.

```
if x < 0:
    pass    # TO DO: handle negative values!
```

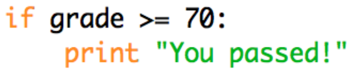
Example Problem

In class, we translate the flowchart from Quiz 1 into a Python program.

Example: If

Write a program which asks a user for their grade. If it is above or equal to 70, tell them they passed.

Solution



```
if grade >= 70:
    print "You passed!"
```

if-else

It is also possible to specify what will happen if the if condition is not satisfied, using **else**. The syntax is as follows:

```
if condition:
    statement

else:
    statement
```

if-else

If `test_next_week` is *not* true, the `else` condition is executed.

```
def main():
    test_next_week = True

    if test_next_week:
        print "There is a test next week."

    else:
        print "There is not a test next week."

main()
```

Exercise: If-Else

Write a program which asks the user for their grade. If their grade is greater than or equal to 70, tell them they passed. Otherwise, tell them they failed.

Solution!

```
if grade >= 70:
    print "You passed!"

else:
    print "You failed :("
```

Nested Decision

We may also have *nested decision structures*, or decision structures inside of decision structures.

```
if hungry_hippo == True:
    if very_hungry_hippo == True:
        print "Eat 10 marbles!"

    else: # AKA, hungry but not very_hungry
        print "Eat 1 marble!"

else: # AKA, if hippo is not hungry
    print "No marbles for you!"
```

Nested Decision

They may be nested arbitrarily deep.

```
if grade >= 70:
    if grade >= 80:
        if grade >= 90:
            print "You got an A!"

        else:
            print "You got a B"
    else:
        print "You got a C!"

else:
    print "Sorry, you failed."
```

Nested Decision vs. if-elif-else

The previous slide's example runs correctly, but looks messy. We can rewrite nested decision structures using `elif`, which stands for "else, if."

Nested decision with if-elif-else

`elif` allows us to consider multiple possibilities, and has syntax as follows:

```
if condition:
    statement

elif condition:
    statement

else:
    statement
```

Example: if-elif-else

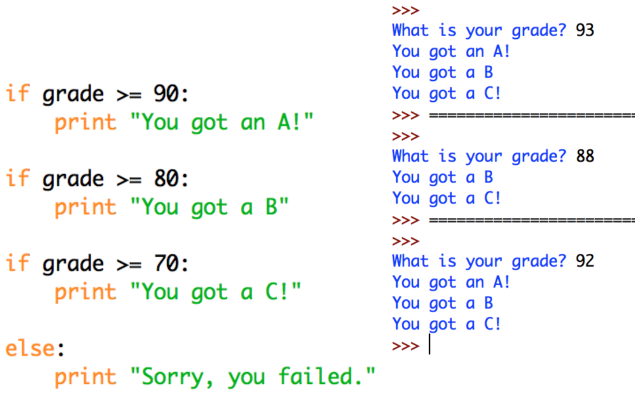
Exercise: Rewrite the nested grades example using `elif` (with no nesting.)

Solution!

```
if grade >= 90:
    print "You got an A!"

elif grade >= 80:
    print "You got a B"

elif grade >= 70:
    print "You got a C!"

else:
    print "Sorry, you failed."
```

Why elif?

What if we wrote the previous example using only `if`, and no `elif` statements?



```
if grade >= 90:
    print "You got an A!"

if grade >= 80:
    print "You got a B"

if grade >= 70:
    print "You got a C!"

else:
    print "Sorry, you failed."
```

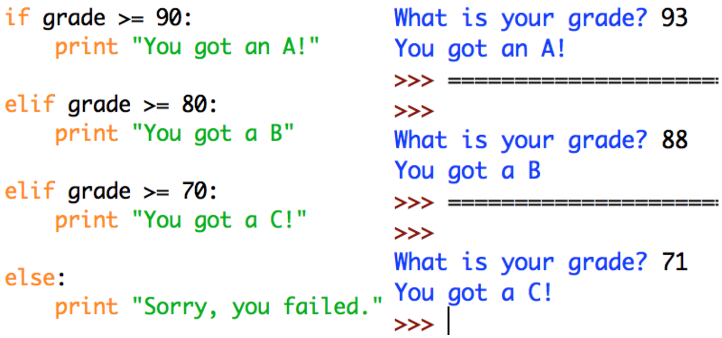
```
>>>
What is your grade? 93
You got an A!
You got a B
You got a C!
>>> ========================
>>>
What is your grade? 88
You got a B
You got a C!
>>> ========================
>>>
What is your grade? 92
You got an A!
You got a B
You got a C!
>>> |
```

Figure: Output with if statements for various grades

Why elif?

The `elif` statement can be interpreted as “otherwise, if..." It forces the cases to be *mutually exclusive.* The grades example does not work without `elif` because the conditions are not mutually exclusive. However, using `elif` forces the cases to be mutually exclusive.

Why elif?



```
if grade >= 90:
    print "You got an A!"

elif grade >= 80:
    print "You got a B"

elif grade >= 70:
    print "You got a C!"

else:
    print "Sorry, you failed."
```

```
What is your grade? 93
You got an A!
>>> ======================
>>>
What is your grade? 88
You got a B
>>> ======================
>>>
What is your grade? 71
You got a C!
>>> |
```

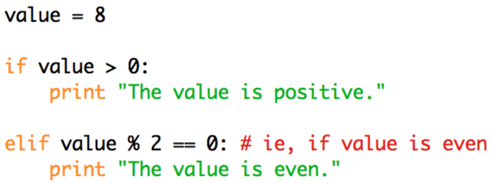
Figure: Output with elif for various grades

Cases which shouldn’t be mutually exclusive

Decide if you want your cases to be mutually exclusive or not before deciding upon a decision structure.

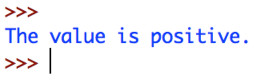
Mutually exclusive?

Consider the following example, which tests whether a value is positive and whether it is even.



```
value = 8

if value > 0:
    print "The value is positive."

elif value % 2 == 0: # ie, if value is even
    print "The value is even."
```

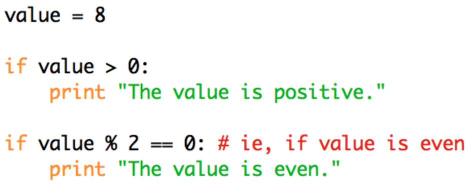
The value 8 is both positive and even. However when we run the program, the output is:



```
>>>
The value is positive.
>>> |
```

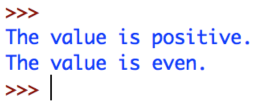
Mutually exclusive? No.

The `elif` structure is for *mutually exclusive* cases, but a number can be **both** positive and even! Rewrite as follows



```
value = 8

if value > 0:
    print "The value is positive."

if value % 2 == 0: # ie, if value is even
    print "The value is even."
```

to receive output



```
>>>
The value is positive.
The value is even.
>>> |
```

Mix and match!

We can combine nested decision structures and `elif` to create even more complicated algorithms. The following two algorithms are, in fact, equivalent; one uses *nesting* while the other uses `elif`.

```
if value > 0:
    print "The value is positive."

elif value == 0:
    print "The value is zero."

else:
    print "The value is negative."

if value % 2 == 0:
    print "The value is even."

else:
    print "The value is odd."
```

```
if value >= 0:
    if value == 0:
        print "The value is zero."

    else:
        print "The value is positive."

else:
    print "The value is negative."

if value % 2 == 0:
    print "The value is even."

else:
    print "The value is odd."
```

Example: Converting from nested to elif

Convert the following program to `elif` structure; in other words, rewrite it so that there are no nested conditions!

```
def main():
    temp = input("What is the temperature? ")
    rain = input("Is it raining? (Type True for yes, False for no) ")

    if temp >= 80:
        if rain == True:
            print "It's hot and rainy."

        else:
            print "It's hot and dry."

    else:
        if rain ==  True:
            print "It's not too hot, but it's raining."

        else:
            print "It's not hot and not raining."

main()
```

Solution!

```
def main():
    temp = input("What is the temperature? ")
    rain = input("Is it raining? (Type True for yes, False for no) ")

    if temp >= 80 and rain == True:
        print "It's hot and rainy."

    elif temp >= 80 and rain == False:
        print "It's hot and dry."

    elif temp < 80 and rain == True:
        print "It's not too hot, but it's raining."

    else:
        print "It's not hot and not raining."

main()
```

An Observation

Note that the examples we have used so far have been naive, in that they assume the user inputs their answer as the correct data type, in the correct range. (For instance, if the user set `rain` equal to `7` in the previous example, what would happen?)

Exercise: Working with the user

Write a program which asks the user to write a positive integer. If the user writes something which is not a positive number of data type `int`, display the message “I’m sorry, incorrect input."

Solution

```
# This program asks the user to choose a positive integer
# and tells them their input was incorrect if they choose something
# which is not a positive int.

def main():
    # Ask the user for a positive integer.

    value = input("Please input a positive integer: ")

    # If their input is a positive integer:
    if type(value) == int and value > 0:
        print "Congratulations, you can follow instructions."

    else:
        print "I'm sorry, incorrect input."

main()
```

Further reading

- How to think like a computer scientist: Learning with Python, chapter 4 `http://www.openbookproject.net/thinkcs/python/english2e/ch04.html`