Math 246 Unit 3A. Making choices

Professor Brenton LeMesurier, September 8, 2015

To move beyond a simple sequence of calculations, two main tools are needed:

- the ability to choose between different actions depending on information like the current values of variables, and
- *iteration*, or the *repetition* of a sequence of actions, with possible variation due to the values of variables being changed during the repetitions.

We consider choices or decision making here, and then iteration in part 3B.

Basic decision-making with `if`

We have already seen the need for choosing between alternatives when dealing with the various cass than can arise in solving a quadratic equation.

The simplest form is doing something if and only if a certain statement is true.

To illustrate this, let us use the integer division remainder operation `%` to decide whether a given integer is even.

In [ ]:

```
n= eval(input('Enter an integer: '))
if n % 2 == 0:
    print(n, 'is even')
```

Arithmetic comparisons

In mathematical applications, decisions are often based on equalities and inequalities, and we have just seen the slightly surprising notation for one of these: asking about equality. Since the Python assignment statement

```
x = y
```

causes the value of y to be copied into the variable x, the equal sign cannot be used like this to ask whether these two are equal (they always are after the copying!). So instead, a double equal sign is used:

```
x == y
```

Here is the complete list of arithmetic comparisons:

Equality and inequality are stated with

- `x == y` to assert that $x = y$
- `x != y` to assert that $x \neq y$

The strict inequality signs are actually on the keyboard, so we have

- `x < y`
- `x > y`

For non-strict inequalities, we append an equal sign

- `x <= y` to assert that $x \leq y$
- `x >= y` to assert that $x \geq y$

Logical (or "Boolean") expressions and their two possible values, "True" and "False"

Let us look at just that True/False statement in the test for evenness above -- and add a test for oddness too:

In [ ]:

```
n = eval(input('Enter an integer: '))
print(n % 2 == 0)
print(n % 2 != 0)
```

These logical expressions have the two possible values "`True`" and "`False`", and we can combine statments with logical negation, "and" and "or". Python uses standard words for these, rather than notations like or ! for logical negation, '&' or '&&' for "and", and '|' or '||' for "or", as seen in some programming languages.

In [ ]:

```
n = eval(input('Enter a natural number n, greater than or equal to 0: '))
if n < 0:
    print("Warning: that's not a natural number!  But I will try anyway.")

niseven = n % 2 == 0
nisodd = not niseven
nissingledigit = n < 10
nisateen = n >= 13 and n <= 19
# We can run inequalties together so long they "all point the same way".
nisateen_alt = 13 <= n <= 19
nisnottwodigits = n < 10 or n >= 100
if niseven:
    print(n, 'is even')
if nisodd:
    print(n, 'is odd')
if nissingledigit:
    print(n, 'is a single digit number')
if nisateen:
    print(n, 'is in the range from 13 to 19')
if nisateen_alt:
    print('As I just said,', n, 'is in the range from 13 to 19')
if nisnottwodigits:
    print(n, 'is is not a two digit number')
```

Covering both cases: the optional `else` clause

Sometimes you want to execute one list of commands if a statement is true and another when it is false; in fact we saw that above with the even and odd tests. We can list the commands for the second alternative by adding an optional `else` clause, and thus avoid redundantly making the same test twice:

In [ ]:

```
n= eval(input('Enter an integer: '))
if n % 2 == 0:
    print(n, 'is even')
else:
    print(n, 'is odd')
```

Testing multiple options with `elif` clauses

Sometimes there are more than two mutually exclusive options, and we would like to test for each in turn until we first the right one, and then act on it, but then not waste effort checking the remaining items on the list. For this we use the last optional feature of the `if` statement: the `elif` clause, which basically means "if none of the above were true, but this one is, then do the following".

In [ ]:

```
n= eval(input('Enter an integer: '))
if n % 4 == 0:
    print(n, 'is a multiple of 4')
elif  n % 2 == 0:
    # A multiple of 2 but not of 4, or we would have stoped with the above "hit"
.
    print(n, 'is an odd multiple of 2')
else:
    print(n, 'is odd')
```

There can be as many `elif` clauses as you want, and there may or may not be a final catch-all `else` clause after them. If there is also an `else` clause, it comes last, after all the `elif` clauses.

In [ ]:

```
n= eval(input('Enter an integer: '))
if n % 6 == 0:
    print(n, 'is a multiple of 2 and of 3')
elif  n % 2 == 0:
    print(n, 'is a multiple of 2 but not of 3')
elif  n % 3 == 0:
    print(n, 'is a multiple of 3 but not of 2')
# Note: I am totally uninterested in numbers that are not multiples of two or of
three!
```

Exercise 3.1

Write a Python file that asks for input of three real numbers a, b, and c, and computes and prints all solutions of the equation $ax^2 + bx + c = 0$. Identify and handle all the "edge" cases!