conditionals and flow control (week 2)

Ben Bolker

10:23 20 January 2017

Conditionals and flow control

- **Conditionals**: Do something *if* something else is true
- **Flow control**: Go to different places in the code: especially, repeat calculations
- Everything we need for interesting programs ("the rest is commentary")
- Technically we can compute *anything*: Turing machines (xkcd)

Conditionals

- Do something *if* something is true
- `if` statement (reference)

```
if False:
    print("no")
```

- else-if (`elif`) and `else` clauses

```
if (x<=0):
    print("what??")
elif(x==1):
    print("one")
elif(x==2):
    print("two")
else:
    print("many")
```

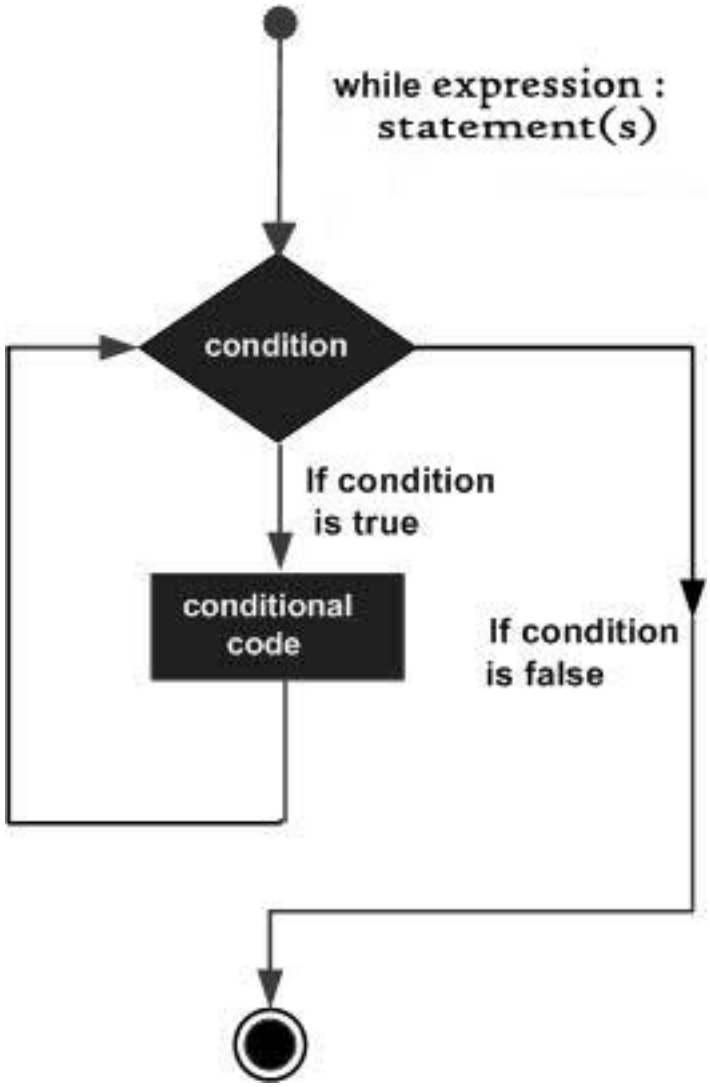
- not too much else to say
- we can do more than one thing; use a *code block*
- indentation is crucial

examples:

- CodingBat date_fashion problem
- CodingBat alarm clock problem

while

- repeat code many times, *while* some logical statement is true (reference)



For example:

```
x = 17
while x>1:
    x = x/2
```

Maybe we want to know how many steps that took:

```
x = 17
n = 0
while x>1:
    x = x/2
    n = n+1
```

- **What is the answer?**
- Can you get the same answer using `import math` and `math.log(x,2)` (and maybe `round()` or `math.floor`)?
- We can use logical operators to combine

```
x = 17
n = 0
while x>1 and n<3:
```

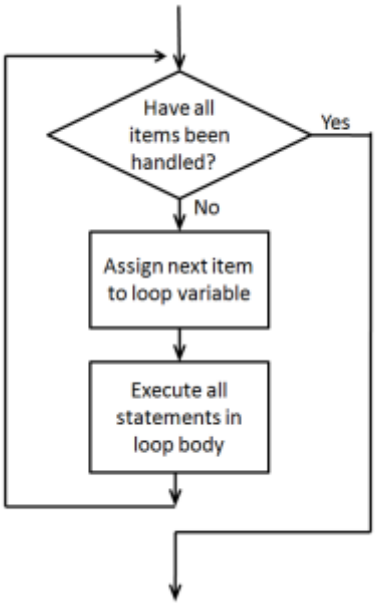


Figure 1: for loop

```
    x = x/2
    n = n+1
```

for loops

- what if we want to repeat a fixed number of times? We could use something like

```
n = 0
while n<n_max:
    # do stuff
    n = n+1
```

Or we could use a `for` loop:

```
for n in range(0,n_max):
    # do stuff
```

- does this repeat `n_max` or `n_max+1` times? (hint: try it out, and/or use `list(range(...))` ...)
- more generally, we can use `for` to iterate over *any list*.

`for` loop examples

- CodingBat > string-2 > countHi
- CodingBat > string-2 > catDog
- CodingBat > Array-2 > bigDiff

Another example: a change-writing program.

Given an amount of money, return a list of length 5 that gives the (smallest) number of coins of each unit (toonies, loonies, quarters, dimes, and nickels) required to make up that amount.

```
total=5.73
toonies = 5.73 // 2 ## integer division
total = total - 2*toonies
```

```
total = 5.73
res = []   # empty list
denoms = list(2,1,0.25,0.1,0.05)
for d in denoms:
    # do stuff
```

- start with `total`, use `denoms` above

1. program to see how many pennies are left (how could we do this much more easily?)
2. **or** print out change as we go along
3. **or** save results as an array

Now let's look at the prime walk program again ...

More CodingBat examples:

- List-2 > count_evens
- List-2 >sum13
- List-2 > bigdiff
- reverse a list (not using slicing)?

`break`

`break` is a way to get out of a `while` or `for` loop early:

```
for i in range(0,10):
   if i>5:
      break
```

nested `for` loops

We can look at (e.g.) all the combinations of `i` and `j` via:

```
for i in range(0,3):
   for j in range(0,3):
      print([i,j])
```

Loops and indices

From Secret Weblog: all of the following are equivalent ...

```
i = 0
while i < mylist_length:
   do_something(mylist[i])
   i += 1  ## or i=i+1
```

vs.

```
for i in range(mylist_length):
   do_something(mylist[i])
```

(this form is useful if we need to combine two lists, or otherwise index element `i` of several different things . . .)

vs.

```
for element in mylist_length:
   do_something(element)
```

Criteria:

- speed
- memory use
- simplicity (code length)
- simplicity (avoid modules)
- simplicity (avoid abstractions)
- pythonicity