Beyond Sequences: The while-loop

**while <condition>:**
- statement 1 
- ... 
- statement n
  - **loop condition**
  - **loop body**

**condition**
- use
- body
- false

- Broader notion of loop
  - You define “more to do”
  - Not limited sequences
- Must manage loop var
  - You create it before loop
  - You update it inside loop
  - For-loop automated it
- Trickier to get right

**Vs For-Loop**

**For-Loop**

```python
def sum_squares(n):
    '''Rets: sum of squares
    Prec: n is int > 0'''
    total = 0
    for x in range(n):
        total = total + x*x
```

**While-Loop**

```python
def sum_squares(n):
    '''Rets: sum of squares
    Prec: n is int > 0'''
    total = 0
    x = 0
    while x < n:
        total = total + x*x
        x = x+1
```

**Tracing While-Loops**

```python
print('Before while')
total = 0
x = 0
while x < n:
    print('Start loop ' + str(x))
    total = total + x*x
    x = x + 1
print('End loop ')
print('After while')
```

**Output:**

```
Before while
Start loop 0
End loop
Start loop 1
End loop
Start loop 2
End loop
After while
```

**How to Design While-Loops**

- Many of the same rules from for-loops
  - Often have an accumulator variable
  - Loop body adds to this accumulator
- Differences are loop variable and iterable
  - Typically do not have iterable
- Breaks up into three design patterns
  1. Replacement to range()
  2. Explicit goal condition
  3. Boolean tracking variable

**Replacing the Range Iterable**

```
range(a,b)
```

```
range(c,d+1)
```

**Using the Goal as a Condition**

```python
def prompt(prompt,valid):
    '''Returns: the choice from a given prompt.
    Preconditions: prompt is a string, valid is a tuple of strings'''
    response = input(prompt)
    # Continue to ask while the response is not valid.
    while not (response in valid):
        print('Invalid response. Answer must be one of ') + str(valid)
        response = input(prompt)
    return response
```
Using a Boolean Variable

```python
def roll_past(goal):
    #**Returns: The score from rolling a die until passing goal.**
    loop = True # Keep looping until this is false
    score = 0
    while loop:
        roll = random.randint(1,6)
        if roll == 1:
            score = 0; loop = False
        else:
            score = score + roll; loop = score < goal
    return score
```

Advantages of while vs for

- **Table of squares to N**
  ```python
  seq = []
  n = floor(sqrt(N)) + 1
  for k in range(n):
      seq.append(k*k)
  ```
  - A for-loop requires that you know where to stop the loop ahead of time
  - A while loop can use complex expressions to check if the loop is done

Difficulties with while

- Be careful when you modify the loop variable
  ```python
def rem3(lst):
    #**Remove all 3's from lst**
    i = 0
    while i < len(lst):
        # no 3's in lst[0..i-1]
        if lst[i] == 3:
            del lst[i]
        else:
            i = i+1
  ```
  - The stopping condition is not a numerical counter this time.
  - Simplifies code a lot.

Application: Convergence

- **How to implement this function?**
  ```python
def sqrt(c):
    #**Returns the square root of c**
    x = c/2.0
    while abs(x*x-c) > err:
        # Get x_{n+1} from x_n
        x = x/2.0+c/(2.0*x)
    return x
  ```
  - Value sqrt(c) is a root of this polynomial
  - Suggests a use for Newton’s Method
  - Start with a guess at the answer
  - Use calculus formula to improve guess

The Final Result

```python
def sqrt(c,err=1e-6):
    #**Returns: sqrt of c with given margin of error.**
    x = c/2.0
    while abs(x*x-c) > err:
        # Get x_{n+1} from x_n
        x = x/2.0+c/(2.0*x)
    return x
```

Using while-loops Instead of for-loops

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Better for modifying data</td>
<td>Performance is slower</td>
</tr>
<tr>
<td>More natural than range</td>
<td></td>
</tr>
<tr>
<td>Works better with deletion</td>
<td>Python optimizes for-loops</td>
</tr>
<tr>
<td>Better for convergent tasks</td>
<td>Cannot optimize while</td>
</tr>
<tr>
<td>Loop until calculation done</td>
<td>Infinite loops more likely</td>
</tr>
<tr>
<td>Exact steps are unknown</td>
<td>Easy to forget loop vars</td>
</tr>
<tr>
<td>Easier to stop early</td>
<td>Or get stop condition wrong</td>
</tr>
<tr>
<td>Just set loop var to False</td>
<td>Debugging is harder</td>
</tr>
<tr>
<td></td>
<td>Will see why in later lectures</td>
</tr>
</tbody>
</table>