Beyond Sequences: The **while**-loop

<table>
<thead>
<tr>
<th>while &lt;condition&gt;:</th>
</tr>
</thead>
<tbody>
<tr>
<td>loop body</td>
</tr>
<tr>
<td>statement 1</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>statement n</td>
</tr>
</tbody>
</table>

**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*

**While Versus for**

<table>
<thead>
<tr>
<th>For-Loop</th>
<th>While-Loop</th>
</tr>
</thead>
<tbody>
<tr>
<td>def sum_squares(n):</td>
<td></td>
</tr>
<tr>
<td>&quot;<strong>Rets: sum of squares</strong>&quot;</td>
<td></td>
</tr>
<tr>
<td>Free: n is int &gt; 0**&quot;**</td>
<td></td>
</tr>
<tr>
<td>total = 0</td>
<td></td>
</tr>
<tr>
<td>for x in range(n):</td>
<td></td>
</tr>
<tr>
<td>total = total + x*x</td>
<td></td>
</tr>
<tr>
<td>while x &lt; n:</td>
<td></td>
</tr>
<tr>
<td>total = total + x*x</td>
<td></td>
</tr>
<tr>
<td>x = x+1</td>
<td></td>
</tr>
</tbody>
</table>

**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**

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

```python
i = a
while i < b:
    process integer i
    i = i + 1
```

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

```python
i = c
while i <= d:
    process integer i
    i = i + 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

```python
# table of squares to N
seq = []
n = floor(sqrt(N)) + 1
for k in range(n):
    seq.append(k*k)
```

```python
# table of squares to N
seq = []
k = 0
while k*k < N:
    seq.append(k*k)
    k += 1
```

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

```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 += 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
```

Better for modifying data:
- More natural than range
- Works better with deletion
- Better for convergent tasks
- Loop until calculation done
- Exact steps are unknown
- Easier to stop early
- Just set loop var to False

Performance is slower:
- Python optimizes for-loops
- Cannot optimize while
- Infinite loops more likely
- Easy to forget loop vars
- Or get stop condition wrong
- Debugging is harder
- Will see why in later lectures

The Final Result

```python
def sqrt(c,err=1e-6):
    """Returns: sqrt of c with given margin of error.
    Preconditions: c and err are numbers > 0""
    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
```