

Beyond Sequences: The while-loop

```

while <condition>:
    statement 1
    ...
    statement n
            
```

Vs For-Loop

- 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

1

while Versus for

For-Loop	While-Loop
<pre> def sum_squares(n): """Rets: sum of squares Prec: n is int > 0""" total = 0 for x in range(n): total = total + x*x </pre>	<pre> 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 </pre>

Must remember to increment

2

Tracing While-Loops

```

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')
            
```

Important

Output:

```

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

Important

3

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

4

Replacing the Range Iterable

range(a,b)	range(c,d+1)
<pre> i = a while i < b: process integer i i = i + 1 </pre>	<pre> i = c while i <= d: process integer i i = i + 1 </pre>
<pre> # store in count # of '/'s in String s count = 0 i = 0 while i < len(s): if s[i] == '/': count = count + 1 i = i + 1 # count is # of '/'s in s[0..s.length()-1] </pre>	<pre> # Store in double var. v the sum # 1/1 + 1/2 + ... + 1/n v = 0; # call this 1/0 for today i = 1 while i <= n: v = v + 1.0 / i i = i + 1 # v = 1/1 + 1/2 + ... + 1/n </pre>

5

Using the Goal as a Condition

```

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
            
```

6

Using a Boolean Variable

```
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
```

Track the condition

7

Advantages of while vs for

# table of squares to N	# table of squares to N
seq = []	seq = []
n = floor(sqrt(N)) + 1	k = 0
for k in range(n):	while k*k < N:
seq.append(k*k)	seq.append(k*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

8

Difficulties with while

Be careful when you **modify** the loop variable

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	def rem3(lst): """Remove all 3's from lst""" while 3 in lst: lst.remove(3)
---	---

Stopping point keeps changing

The stopping condition is not a numerical counter this time. Simplifies code a lot.

9

Application: Convergence

- How to implement this function?

```
def sqrt(c):
    """Returns the square root of c"""
```
- Consider the polynomial $f(x) = x^2 - c$
 - Value $\text{sqrt}(c)$ 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

10

The Final Result

```
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
```

11

Using while-loops Instead of for-loops

Advantages

- 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

Disadvantages

- 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

12