

### Example: Summing the Elements of a List

```
def sum(thelist):
    """Returns: the sum of all elements in thelist
    Precondition: thelist is a list of all numbers
    (either floats or ints)"""
    result = 0
    result = result + thelist[0]
    result = result + thelist[1]
    ...
    return result
```

There is a problem here

### Working with Sequences

- Sequences are potentially **unbounded**
  - Number of elements inside them is not fixed
  - Functions must handle sequences of different lengths
  - Example:** sum([1,2,3]) vs. sum([4,5,6,7,8,9,10])
- Cannot process with **fixed** number of lines
  - Each line of code can handle at most one element
  - What if # of elements > # of lines of code?
- We need a new **control structure**

### The For-Loop

# Create local var x	# Write as a for-loop
x = seqn[0]	for x in seqn:
print(x)	print(x)
x = seqn[1]	
print(x)	
...	
x = seqn[len(seqn)-1]	
print(x)	

Not valid Python

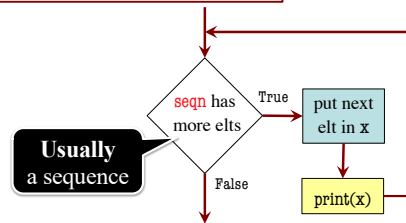
#### Key Concepts

- iterable:** seqn
- loop variable:** x
- body:** print(x)

### Executing a For-Loop

```
The for-loop:
for x in seqn:
| print(x)
```

- iterable:** seqn
- loop variable:** x
- body:** print(x)



### Example: Summing the Elements of a List

```
def sum(thelist):
    """Returns: the sum of all elements in thelist
    Precondition: thelist is a list of all numbers
    (either floats or ints)"""
    result = 0
    for x in thelist:
        result = result + x
    return result
```

Accumulator variable

- iterable:** thelist
- loop variable:** x
- body:** result=result+x

### Example: String-Based Accumulator

```
def despace(s):
    """Returns: s but with its spaces removed
    Precondition: s is a string"""
    result = ""
    for x in s:
        if x != " ":
            result = result+x
    return result
```

Body

### Modifying the Contents of a List

```
def add_one(thelist):
    """(Procedure) Adds 1 to every element in the list
    Precondition: thelist is a list of all numbers
    (either floats or ints)"""
    for x in thelist:
        x = x+1
    # procedure; no return
```

**DOES NOT WORK!**

### On The Other Hand

```
def copy_add_one(thelist):
    """Returns: copy with 1 added to every element
    Precondition: thelist is a list of all numbers
    (either floats or ints)"""
    mycopy = [] # accumulator
    for x in thelist:
        x = x+1
        mycopy.append(x) # add to end of accumulator
    return mycopy
```

Accumulator keeps result from being lost

### How Can We Modify A List?

- **Never** modify loop var!
- This is an infinite loop:
- Need a second sequence
- How about the *positions*?

```
for x in thelist:
    thelist.append(1)
```

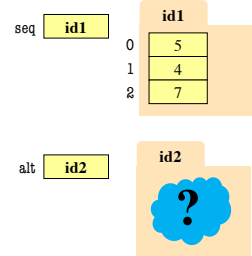
```
thelist = [5, 2, 7, 1]
thepos = [0, 1, 2, 3]
```

Try in Python Tutor to see what happens

```
for x in thepos:
    thelist[x] = x+1
```

### This is the Motivation for Iterables

- **Iterables** are objects
  - Contain data like a list
  - **But cannot slice them**
- Have list-like properties
  - Can use them in a for-loop
  - Can convert them to lists
  - `mylist = list(myiterable)`
- **Example:** Files
  - Use `open()` to create object
  - Makes iterable for reading



### The Range Iterator

- `range(x)`
  - Creates an iterator
  - Stores `[0,1,...,x-1]`
  - **But not a list!**
  - But try `list(range(x))`
- `range(a,b)`
  - Stores `[a,...,b-1]`
- `range(a,b,n)`
  - Stores `[a,a+n,...,b-1]`
- Very versatile tool
- Great for processing ints

```
total = 0
# add the squares of ints
# in range 2..200 to total
for x in range(2,201):
    total = total + x*x
```

Accumulator

### Modifying the Contents of a List

```
def add_one(thelist):
    """(Procedure) Adds 1 to every element in the list
    Precondition: thelist is a list of all numbers
    (either floats or ints)"""
    size = len(thelist)
    for k in range(size):
        thelist[k] = thelist[k]+1
    # procedure; no return
```

Iterator of list positions (safe)

**WORKS!**