Iterators: Iterables Outside of For-Loops

- Iterators can **manually** extract elements
  - Get each element with the `next()` function
  - Keep going until you reach the end
  - Ends with a `StopIteration` (Why?)
- Can create iterators with `iter()` function
  ```python
  >>> a = iter([1,5,3])
  >>> next(a)
  1
  >>> next(a)
  5
  ```

Iterators are Classes

- Has the same problem as GUI applications
  - We have a hidden loop
  - All loop variables are now attributes
  - Similar to inter-frame/intra-frame reasoning
- Would be easier if loop were **not** hidden
  - **Idea:** Write this as a function definition
  - Function makes loop/loop variables visible
- But iterators “return” multiple values
  - So how would this work?

Iterators are Hard to Write!

```python
def range2iter(n):
    """Generator for the squares of numbers 0 to n-1"
    Generator for the squares of numbers 0 to n-1
    Precon: n is an int >= 0
    for x in range(n):
        yield x*x
```

The `yield` Statement

- **Format:** `yield <expression>`
  - Used to produce a value
  - But it **does not stop** the “function”
  - Useful for making iterators
- **But:** These are not normal functions
  - Presence of a yield makes a **generator**
  - Function that returns an iterator
What Happens on a Function Call?

- Function input is an iterable (string, list, tuple)
- Function output typically a transformed copy
- **Old way:** Accumulate a new list or tuple
- **New way:** Yield one element at a time

Generators Are Easy

- New way makes an **iterator** (not iterable)
  - So can only be used once!
  - But easily turned into a list or tuple

Generators: The New Way

```python
def add_one(input):
    # Generates 1 added to each element of input
    Precond: input is an iterable of all numbers
    for x in input:
        yield x + 1
```

Chaining Generators

- Generators can be chained together
  - Take an iterator/iterable as input
  - Produce an iterator as output
  - Output of one generator = input of another
- Powerful programming technique

Accumulators: The Old Way

```python
def add_one(lst):
    # Returns copy with 1 added to every element
    copy = []
    for x in lst:
        x = x + 1
        copy.append(x)
    return copy
```