For Loops

Thomas Schwarz, SJ

Preview : Lists

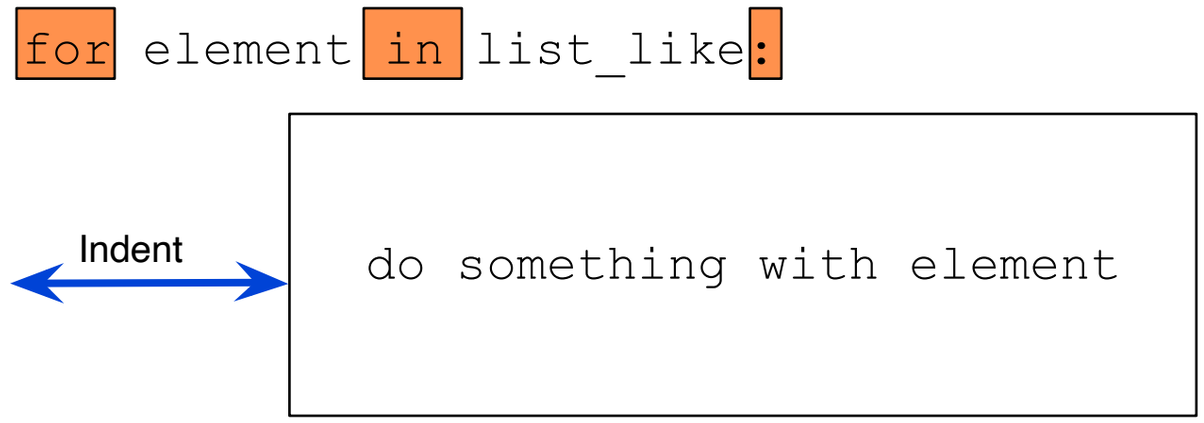
- A list is an ordered collection of elements
  - Uses brackets [ ]
  - An empty list `[ ]`
  - A list with one element `[2]`
  - A list with two elements `[2,3]`

Preview : Lists

- Python lists can have elements of different types
  - `my_list = [1, 1.0, 'one']`

For statement

- A for statement does something for everything in a list-like structure



- Key word is for
- element is a variable that takes on repetitively all elements of the list
- List-like is anything that is like a list (e.g. a list)

For statement

- Example:
  - A table to convert from rupees to euros

```
RUPEES_TO_EUROS = 0.01184638023
for rupees in [1, 2, 5, 10, 20, 50, 100, 200, 500, 1000, 2000]:
    euros = rupees*RUPEES_TO_EUROS
    print(rupees, '(IR) corresponds to', round(euros, 2), '€')
```

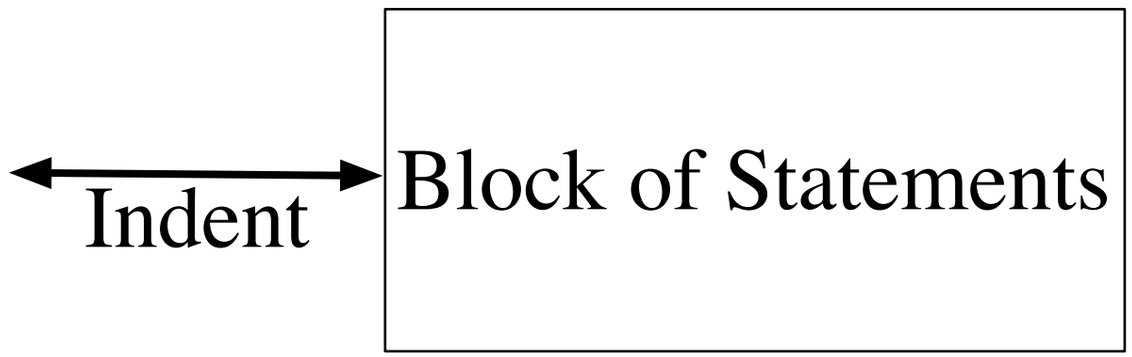
For statement

- Many times we just want consecutive numbers
  - These are generated through range
    - Python 1,2: Range generates a list
    - Python 2: xrange generates next number on demand
    - Python 3: range gives a “range” object
- Example for efficiency can be the enemy of ease of understanding
- For the time being: a range is just a list of numbers

For statement

- For statement with range:

```
for i in range(n):
```



- Range is equivalent to a list [0, 1, …, n-1]

For statement

- Why do we start with 0?
  - In early CS, for loops were used over offsets into memory
  - Better start with 0 then to not have to distinguish between a memory region and the preceding memory location
  - And range(n) should have n elements, so we stop with n-1

For statement

- Example:
  - Calculate the sum $\sum_{i=0}^{1000} \frac{1}{1+i^2}$

- Need an **accumulator** to keep the partial sums
  - This accumulator needs to be initialized to 0
  - We update this accumulator by adding $\frac{1}{1+i^2}$ to it repeatedly
    - Can use the += statement

For statement

- `x += 7`
  - is a shortcut for
- `x = x+7`
- Same goes for `x/= 2, x*=2, x-=2`

For statement

- Notice that the last value for *i* is 1000, therefore we need `range(1001)`
  - Refer to 1001 as the *stop value*

For statement

- First draft of code:

```
accumulator = 0
for i in range(1001):
    accumulator += 1/(1+i**2)
print(accumulator)
```

- Disregards principles of numerical analysis
  - First addend is 1
  - Second addend is 1/5
  - …
  - Last addend is 1/1,000,001

For statement

- First draft of code

```
accumulator = 0
for i in range(1001):
    accumulator += 1/(1+i**2)
print(accumulator)
```

- We are adding small numbers to big numbers
  - Can expect loss of precision

Ranges

- Ranges are quite flexible
  - If there is a single variable, then that variable is the *stop value*
  - If there are two variables, then the first is the *start value*
    - and the second the stop value
  - If there are three variables, then the first one is the start value
    - the second the stop value
    - and the third the *stride*

Ranges

- Example:

```
for i in range(3):
    print(i)
```

- prints

```
0
1
2
```

Ranges

- Example:

```
for i in range(1, 4):
    print(i)
```

- prints

```
1
2
3
```

- Start value is 1, stop value is 4, i.e. last value is 3

Ranges

- Strides:
  - What gets added to the loop variable
- Example

```
for i in range(2,9,3):
    print(i)
```

  - First value for `i` is the start value 2
    - print out 2
  - Then add 3 (the stride to the current value of `i`)
    - print out 5
  - Then add 3 again
    - print out 8
  - Then add 3 again
    - This gives `i=11,` which is ≥ 9, the stop value, so we stop the loop

Ranges

- Example:
  - ```
    for i in range(2,9,3):
        print(i)
    ```
- prints out
  - ```
    2
    5
    8
    ```

Ranges

- Strides can be negative
  - Example:
    -
```
for i in range(10,0,-1):
    print(i)
```

```
10
9
8
7
6
5
4
3
2
1
```

- Stop **before** we reach 0, i.e. at 1

Ranges

- Resumen:
  - range takes up to three parameters:
    - Start
    - Stop
    - Stride

For statement

- Now we can calculate the sum in a safer way

```
accumulator = 0
for i in range(1000, -1, -1):
    accumulator += 1/(1+i**2)
print(accumulator)
```

- modifies result slightly using stride -1
- We start with i = 1000
- We finish with i = 0
- Therefore: stop value is -1