Loops and lists

Foundation of programming (CK0030)

Francesco Corona

FdP

- ☻ Intro to variables, objects, modules, and text formatting
- ☹ **Programming with WHILE- and FOR-loops, and lists**
- ☹ Functions and IF-ELSE tests

- ☹ Data reading and writing
- ☹ Error handling
- ☹ Making modules

- ☹ Arrays and array computing
- ☹ Plotting curves and surfaces

Alternative implementations

Loops and lists

Alternative implementations

Usually, there are alternative ways to write code that solves a problem

- We explore alternative constructs and programs
- Store numbers in lists and print out tables

WHILE loops as FOR loops

Alternative implementations

Loops and lists

FC
CK0030
2018.1

Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Traversing nested lists
Some list operations
Tuples

WHILE loops as FOR loops

Definition

Any `FOR-loop` can be implemented as a `WHILE-loop`

Consider the general piece of code

```
for element in somelist:
    <process element>
```

It can be re-written

```
index = 0

while index < len(somelist):
    element = somelist[index]
    <process element>
    index += 1
```

■

Loops and lists

FC
CK0030
2018.1

Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Traversing nested lists
Some list operations
Tuples

Example

Printout of the Celsius-Fahrenheit table of temperatures

```
Cdegrees = [-20,-15,-10,-5,0,5,10,15,20,25,30,35,40]

print '   C      F'

index = 0
while index < len(Cdegrees):
    C = Cdegrees[index]
    F = (9.0/5)*C + 32
    print '%5d %5.1f' % (C, F)
    index += 1
```

Loops and lists

FC
CK0030
2018.1

Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Traversing nested lists
Some list operations
Tuples

```
Cdegrees = [-20,-15,-10,-5,0,5,10,15,20,25,30,35,40]

print '   C      F'

for C in Cdegrees:
    F = (9.0/5)*C + 32
    print '%5d %5.1f' % (C, F)
```

```
C = -20
dC = 5

while C <= 40:
    F = (9.0/5)*C + 32
    print C, F
    C = C + dC
```

■

Range construction
Alternative implementations

Range construction

It is often tedious to manually type the many elements in `Cdegrees`

- We should use a loop to automate the list construction

```
C_value = -50
C_max = 200
Cdegrees = []

while C_value <= C_max:
    Cdegrees.append(C_value)
    C_value += 2.5                              # C_value = C_value + 2.5
```

The **range construction** is a particularly useful tool for the task

Range construction (cont.)

Definition

range(n)

range(n) generates a list of sequential integers in $[0, n-1]$

- *(Integer n is not included)*

⇝ *0, 1, 2, . . . , n-1*

range(start, stop, step) generates a list of integers in a sequence

⇝ *start, start + (1*step), start + (2*step) up to stop*

- *(stop is not included)*

range(start, stop) is the same as range(start, stop, 1)

⇝ *start, start + (1*1), start + (2*1) up to stop*

- *(That is, step = 1)*

Range construction (cont.)

Things to remember

In Python 2.x, function `range(n)` returns a `list object`

In Python 3.x, function `range(n)` returns a `range object`

- A `range object` can be converted to a `list object`

⇝ `list(range(n))`

This exists in Python 2.x as function `xrange(n)`

Range construction (cont.)

Example

Consider the following examples

`range(2, 8, 3)`

- The output

⇝ `2`

⇝ `2 + (1*3) = 5` (but not `8 = 2 + (2*3)`)

`range(1, 11, 2)`

- The output

⇝ `1`

⇝ `3 = 1 + (1*2)`

⇝ `5 = 1 + (2*2)`

⇝ `7 = 1 + (3*2)`

⇝ `9 = 1 + (4*2)`

■

Loops and lists
FC
CK0030
2018.1
Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Traversing nested lists
Some list operations
Tuples

Range construction (cont.)

A `FOR-loop` over the `list` (object) of integers (type `int` objects) from `range`

```
for i in range(start, stop, step):
 ...                                        # Some operation on element
                                            # <Process element>
```

Loops and lists
FC
CK0030
2018.1
Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Traversing nested lists
Some list operations
Tuples

Range construction (cont.)

Example

We use `range` to create a list `Cdegrees` with values `[-20,-15,...,35,40]`

- Two ways (with and without a loop)

```
Cdegrees = []                          # Create empty list to be filled

for C in range(-20, 45, 5):            #     Pick element C from a list
 Cdegrees.append(C)                    #         of sequential integers

                                       # Element C, inside the FOR loop
                                       # 1st element: -20
                                       # 2nd element: -20 + (1*5) = -15
                                       # 3rd element: -20 + (2*5) = -10
                                       # ...
```

```
Cdegrees = range(-20, 45, 5)
```

To include integer 40, the upper limit must be greater than 40

⇝ This is important

■

Loops and lists
FC
CK0030
2018.1
Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Traversing nested lists
Some list operations
Tuples

Range construction (cont.)

Example

Suppose that now we want to create a slightly different `Cdegrees` list

- `[-10, -7.5, -5, ..., 35, 37.5 ,40]`
- The spacing between entries is 2.5
- The entries are real numbers

We cannot use `range` directly, we must adapt its use

⇝ `range(-10, 45, 2.5)` would give an error

⇝ `range` can only create integers

⇝ We have decimal degrees

Range construction (cont.)

We must introduce an **integer counter** `i` generate by function `range`

- We generate `C` values by $C = -10 + \underbrace{i}_{\uparrow} \cdot 2.5,\ i = 0, 1, 2, \ldots, 20$

```
Cdegrees = []

for i in range(0, 21):              # Generate a range of integers
 C = -10 + i*2.5                    #       Element i is used here
 Cdegrees.append(C)
```

■

Loops and lists

FC
CK0030
2018.1

Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Traversing nested lists
Some list operations
Tuples

FOR loops with list indexes

Alternative implementations

Loops and lists

FC
CK0030
2018.1

Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Traversing nested lists
Some list operations
Tuples

FOR-loops with list indexes

Consider an alternative to iterating over (the elements of) a list directly

```
for element in somelist:
 ...                                # Some operation on element
 ...                                #          <Process element>
```

We can iterate over list indices and then index the list inside the loop

```
for i in range(len(somelist)):
 element = somelist[i]
 ...                                # Some operation on element
 ...                                #          <Process element>
```

`len(somelist)` returns the length of `somelist`

⇝ Indices start at `0`, the largest valid index is `len(somelist)-1`

⇝ `range(len(somelist))` is `[0, 1, ..., len(somelist)-1]`

Loops and lists

FC
CK0030
2018.1

Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Traversing nested lists
Some list operations
Tuples

FOR loops with list indexes (cont.)

Iterating over loop indices is often a useful programming practice

- An example is when we need to process two lists
- (At the same time)

FOR loops with list indexes (cont.)

Example

Suppose that we want to create two lists, `Cdegrees` and `Fdegrees`

Then, suppose that we want to use the two lists to write a table

- The table must have `Cdegrees` and `Fdegrees` as columns

```
n = 21
C_min = -10; C_max = 40                              # Min and max value of C
dC = (C_max - C_min)/float(n-1)                      #          Increment in C


Cdegrees = []                                        #        Build the C list
for i in range(0, n):                                #         Initially empty
 C = -10 + i*dC
 Cdegrees.append(C)


Fdegrees = []                                        #        Build the F list
for C in Cdegrees:                                   #         Initially empty
 F = (9.0/5)*C + 32
 Fdegrees.append(F)


for i in range(len(Cdegrees)):                       #  Print the joint table
 C = Cdegrees[i]                                     #       Loop over indexes
 F = Fdegrees[i]                                     #       Loop over indexes
 print '%5.1f %5.1f' % (C, F)
```

Loops and lists

FC
CK0030
2018.1

Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Traversing nested lists
Some list operations
Tuples

FOR loops with list indexes (cont.)

In the example, we started with empty lists then appended new elements

We can start with lists of correct size, containing, say, zeros

- Then, we index the lists to fill in actual values

Loops and lists

FC
CK0030
2018.1

Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Traversing nested lists
Some list operations
Tuples

FOR loops with list indexes (cont.)

Definition

A list of zeros

How to create a list of length n consisting of zeros

```
somelist = [0]*n
```

Loops and lists

FC
CK0030
2018.1

Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Traversing nested lists
Some list operations
Tuples

FOR loops with list indexes (cont.)

Example

```
n = 21
C_min = -10                                   #                      Min value of C
C_max = +40                                   #                      Max value of C
dC = (C_max - C_min)/float(n-1)               #                      Increment in C


Cdegrees = [0]*n                              # Cdegrees must be of correct length
for i in range(len(Cdegrees)):                #             Initially full of zeros
 Cdegrees[i] = -10 + i*dC


Fdegrees = [0]*n                              # Fdegrees must be of correct length
for i in range(len(Cdegrees)):                #             Initially full of zeros
 Fdegrees[i] = (9.0/5)*Cdegrees[i] + 32


for i in range(len(Cdegrees)):
 print '%5.1f %5.1f' % (Cdegrees[i], Fdegrees[i])
```

Modify list elements

Alternative implementations

Loops and lists

FC
CK0030
2018.1

Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Travessing nested lists
Some list operations
Tuples

Modify list elements

Consider some list of temperature values accessible with name `Cdegrees`

Suppose that we want to change the value of each of its elements

- We want to add `5` (degrees)

```
n = 21; C_min = -10; C_max = 40
dC = (C_max - C_min)/float(n-1)

Cdegrees = []
for i in range(0, n):
 C = -10 + i*dC
 Cdegrees.append(C)

for i in range(len(Cdegrees)):
 Cdegrees[i] += 5                    # Adjust the i-th element to be equal
                                     #                  to itself plus five
```

Loops and lists

FC
CK0030
2018.1

Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Travessing nested lists
Some list operations
Tuples

Modify list elements (cont.)

Things that do NOT work

```
for c in Cdegrees:
 c += 5
```

```
...
c = Cdegrees[0]                # Automatically done in a FOR statement
c += 5

c = Cdegrees[1]                # Automatically done in a FOR statement
c += 5
...
```

Variable `c` can only be used to read list elements

⇝ It does not change them

⇝ Only `c` is changed

Things that DO work

Remark

To change a list element, `Cdegrees[i]`, an assignment must be used

```
Cdegrees[i] = ...                        # Change the i-th list element
```

Loops and lists

FC
CK0030
2018.1

Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Travessing nested lists
Some list operations
Tuples

List comprehension

Alternative implementations

List comprehension

'Run thru a list and for each element create a new element in another list

- This is a frequently encountered task

⇝ (`Fdegrees[i]` from `Cdegrees[i]`)

Python has a special compact syntax for this

⇝ **List comprehension**

Loops and lists
FC
CK0030
2018.1
Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Travessing nested lists
Some list operations
Tuples

List comprehension (cont.)

Definition

List comprehension

The general syntax for list comprehension

```
 ...

newlist = [E(e) for e in list]

 ...
```

`E(e)` is some `expression` involving element `e` of list `list`

Loops and lists
FC
CK0030
2018.1
Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Travessing nested lists
Some list operations
Tuples

List comprehension (cont.)

Example

Consider the following code, the tasks should be familiar

```
Cdegrees = [-5+i*0.5 for i in range(n)]            #  List comprehension
                                                   # Build list Cdegrees

Fdegrees = [(9.0/5)*C+32 for C in Cdegrees]        #  List comprehension
                                                   # Build list Fdegrees

C_plus_5 = [C+5 for C in Cdegrees]                 # Build list C_plus_5
```

How does the computation evolve in each case?

What are the elements of the lists?

Loops and lists
FC
CK0030
2018.1
Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Travessing nested lists
Some list operations
Tuples

Travessing multiple lists

Alternative implementations

Travessing multiple lists

Suppose that we want to use lists `Cdegrees` and `Fdegrees` to make a table

- We need to traverse both arrays

A `for element in list` construction is not suitable here

- It extracts elements from one list only

A solution is to use a `FOR-loop` over indices

- So that we can index both lists
- (We silently used this already)

Loops and lists
FC
CK0030
2018.1
Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Traversing nested lists
Some list operations
Tuples

Travessing multiple lists (cont.)

Example

Consider this piece of code for printing a table of temperature values

```
n=21

Cdegrees = [-5+i*0.5 for i in range(n)]              # List comprehension
Fdegrees = [(9.0/5)*C+32 for C in Cdegrees]          # List comprehension

for i in range(len(Cdegrees)):
 print '%5d %5.1f' % (Cdegrees[i], Fdegrees[i])      # Print temperatures
```

Loops and lists
FC
CK0030
2018.1
Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Traversing nested lists
Some list operations
Tuples

Travessing multiple lists (cont.)

It often happens that two or more lists need be traversed simultaneously

Python offers an alternative to the loop over indices

⇝ A special syntax

- The `zip` function

Loops and lists
FC
CK0030
2018.1
Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Traversing nested lists
Some list operations
Tuples

Travessing multiple lists (cont.)

Definition

Zip

Function `zip` turns `n` lists (`list1`, `list2`, `...`) into a single list of `n`-tuples

```
for e1, e2, ... in zip(list1, list2, ...):
                                              # Element e1 from list1
                                              # Element e2 from list2
                                              # ...
```

For each `n`-tuple (`e1`, `e2`, `...`),

- *The first element (`e1`) is from the first list (`list1`)*
- *The second element `e2` is from second list (`list2`)*
- *...*
- *The n-th element `e2` is from second list (`listn`)*

The loop stops when the end of the shortest list is reached

Travessing multiple lists (cont.)

Example

Consider the following code using list comprehension and the zip function

```
n=21
Cdegrees = [-5+i*0.5 for i in range(n)]              # List comprehension
Fdegrees = [(9.0/5)*C+32 for C in Cdegrees]          # List comprehension

for C, F in zip(Cdegrees, Fdegrees):                  # Print temperatures
 print '%5.1f %5.1f' % (C, F)                           #   Use zip
     function
```


Travessing multiple lists (cont.)

The result of the execution of the first part of the code

```
>>> Cdegrees
 [-5.0,
  -4.5,
  -4.0,
   ...
   4.0,
   4.5,
   5.0]

>>> Fdegrees
 [23.0,
  23.9,
  24.8,
   ...
  39.2,
  40.1,
  41.0]
```


Travessing multiple lists (cont.)

The result of the execution of the second part of the code

```
 -5.0  23.0
 -4.5  23.9
 -4.0  24.8
 -3.5  25.7
 -3.0  26.6
 -2.5  27.5
 -2.0  28.4
 -1.5  29.3
 -1.0  30.2
 -0.5  31.1
  0.0  32.0
  0.5  32.9
  1.0  33.8
  1.5  34.7
  2.0  35.6
  2.5  36.5
  3.0  37.4
  3.5  38.3
  4.0  39.2
  4.5  40.1
  5.0  41.0
```


Travessing multiple lists (cont.)

Consider the continuation code using the zip function and list comprehension

```
table = []
table = [[C,F] for C,F in zip(Cdegrees,Fdegrees)]
```

```
>>> table
[[-5.0, 23.0],
 [-4.5, 23.9],
 [-4.0, 24.8],
 [-3.5, 25.7],
 [-3.0, 26.6],
 [-2.5, 27.5],
 [-2.0, 28.4],
 [-1.5, 29.3],
 [-1.0, 30.2],
 [-0.5, 31.1],
 [0.0,  32.0],
 [0.5,  32.9],
 [1.0,  33.8],
 [1.5,  34.7],
 [2.0,  35.6],
 [2.5,  36.5],
 [3.0,  37.4],
 [3.5,  38.3],
 [4.0,  39.2],
 [4.5,  40.1],
 [5.0,  41.0]]
```

Nested lists
Loops and lists

Loops and lists

FC
CK0030
2018.1

Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Travessing nested lists
Some list operations
Tuples

Nested lists

Nested lists are `list objects`, the `list elements` are `list objects`

We use some examples to motivate the need for nested lists

- We shall also illustrate some basic operations

Loops and lists

FC
CK0030
2018.1

Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Travessing nested lists
Some list operations
Tuples

A table as a row or column list
Nested lists

Loops and lists

FC
CK0030
2018.1

Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Travessing nested lists
Some list operations
Tuples

A table as a row or column list

In our table of temperatures, we used a separate list for each table column

⇝ With n columns, we need n `list objects` to handle table data

```
n=21
Cdegrees = [-5+i*0.5 for i in range(n)]
Fdegrees = [(9.0/5)*C+32 for C in Cdegrees]
Kdegrees = [C+273.15 for C in Cdegrees]

table = []
table = [[C,F,K] for C,F,K in zip(Cdegrees,Fdegrees,Kdegrees)]
```

A table as a row or column list

```
>>> table
[[-5.0, 23.0, 268.15],
 [-4.5, 23.9, 268.65],
 [-4.0, 24.8, 269.15],
   ...,  ...,    ...
 [4.0,  39.2, 277.15],
 [4.5,  40.1, 277.65],
 [5.0,  41.0, 278.15]]
```

Loops and lists
FC
CK0030
2018.1
Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Travessing nested lists
Some list operations
Tuples

A table as a row or column list (cont.)

We think of a table as a single entity, not a collection of n columns

- It is natural to use one argument for the whole table

In Python this can be achieved by using a **nested list**

- Each entry in the list is a list itself

Loops and lists
FC
CK0030
2018.1
Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Travessing nested lists
Some list operations
Tuples

A table as a row or column list (cont.)

A `table object` is understood as a list of lists

We can see it as two different cases

- Either it is a list of the row elements of the table
- Or, it is a list of the column elements of the table

Loops and lists
FC
CK0030
2018.1
Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Travessing nested lists
Some list operations
Tuples

A table as a row or column list (cont.)

Example

```
Cdegrees = range(-20, 41, 5)                    # -20, -15, ..., 35, 40
Fdegrees = [(9.0/5)*C + 32 for C in Cdegrees]

table = [Cdegrees, Fdegrees]
```

- ⇝ The table is a list of two columns
- ⇝ Each column is a list of numbers

```
>>> table
 [[ -20,-15, -10,  -5,   0,   5,  10,  15,  20,  25,  30,  35,   40],
  [-4.0,5.0,14.0,23.0,32.0,41.0,50.0,59.0,68.0,77.0,86.0,95.0,104.0]]
```

A table as a row or column list (cont.)

```
>>> table
[[ -20,-15, -10,  -5,   0,   5,  10,  15,  20,  25,  30,  35,   40],
 [-4.0,5.0,14.0,23.0,32.0,41.0,50.0,59.0,68.0,77.0,86.0,95.0,104.0]]
```

With `table[0]`, we access the first element in the table

⇝ (The `Cdegrees` list)

With `table[1]`, we access the first element in the table

⇝ (The `Fdegrees` list)

```
>>> table[0]
[-20, -15, -10, -5, 0, 5, 10, 15, 20, 25, 30, 35, 40]
>>> Cdegrees
[-20, -15, -10, -5, 0, 5, 10, 15, 20, 25, 30, 35, 40]

>>> table[1]
[-4.0, 5.0, 14.0, 23.0, 32.0, 41.0, 50.0, 59.0, 68.0, 77.0, 86.0, 95.0,
      104.0]
>>> Fdegrees
[-4.0, 5.0, 14.0, 23.0, 32.0, 41.0, 50.0, 59.0, 68.0, 77.0, 86.0, 95.0,
      104.0]
```

Loops and lists
FC
CK0030
2018.1
Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Travessing nested lists
Some list operations
Tuples

A table as a row or column list (cont.)

`table[0][2]` is the third element in the first element (which is a list)

```
>>> table[0]
[-20, -15, -10, -5, 0, 5, 10, 15, 20, 25, 30, 35, 40]

>>> table[0][2]
-10
```

That is also `Cdegrees[2]`

■

Loops and lists
FC
CK0030
2018.1
Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Travessing nested lists
Some list operations
Tuples

A table as a row or column list (cont.)

Consider tabular data with rows and columns

- The underlying data are a nested list
- The first index counts the rows
- The second index counts the columns

This is the convention for indexing elements

Loops and lists
FC
CK0030
2018.1
Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Travessing nested lists
Some list operations
Tuples

A table as a row or column list (cont.)

Example

We can construct `table` as a list of `[C, F]` pairs

- The first index will then run over rows `[C, F]`

```
Cdegrees = range(-20, 41, 5)
Fdegrees = [(9.0/5)*C + 32 for C in Cdegrees]

table = []
for C, F in zip(Cdegrees, Fdegrees):
 table.append([C, F])
```

This construction is based on looping through pairs `C` and `F`

- At each pass, we create a list element `[C, F]`
- Then, we append it as last element to `table`

```
Cdegrees = range(-20, 41, 5)
Fdegrees = [(9.0/5)*C + 32 for C in Cdegrees]

table = []
for C, F in zip(Cdegrees, Fdegrees):
 table.append([C, F])
```

```
>>> table
 [[-20, -4.0],
  [-15,  5.0],
  [-10, 14.0],
  [-5,  23.0],
  [0,   32.0],
  [5,   41.0],
  [10,  50.0],
  [15,  59.0],
  [20,  68.0],
  [25,  77.0],
  [30,  86.0],
  [35,  95.0],
  [40, 104.0]]

>>> table[5]
 [5, 41.0]

>>> table[5][1]
 41.0
```

`table[5]` refers to the sixth element in `table`, a `[C, F]` pair

- With `table[5][0]`, we access the `C` value
- With `table[5][1]`, we access the `F` value

Loops and lists
FC
CK0030
2018.1
Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Traversing nested lists
Some list operations
Tuples

A table as a row or column list (cont.)

```
Cdegrees = range(-20, 41, 5)
Fdegrees = [(9.0/5)*C + 32 for C in Cdegrees]

table = []
for C, F in zip(Cdegrees, Fdegrees):
 table.append([C, F])
```

More compactly, we can obtain the same resultby using list comprehension

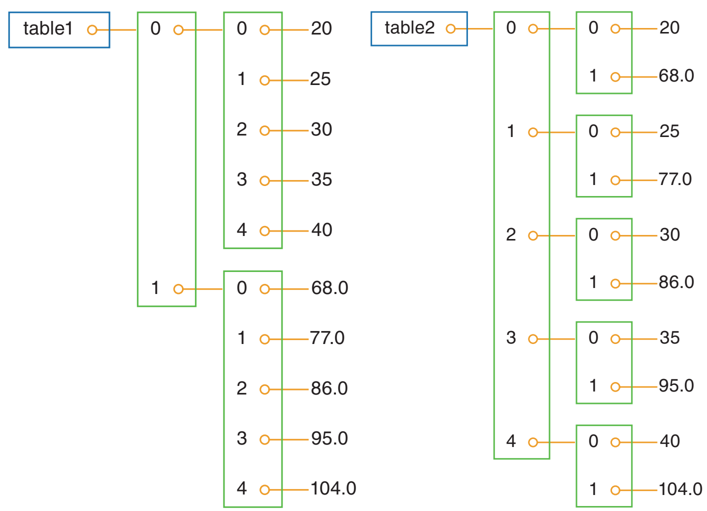
```
Cdegrees = range(-20, 41, 5)
Fdegrees = [(9.0/5)*C + 32 for C in Cdegrees]

table = [[C, F] for C, F in zip(Cdegrees, Fdegrees)]
```

This construction is based on looping through pairs `C` and `F`

- At each pass, we create a list element `[C, F]`
- (The process of appending it not explicit)

Loops and lists
FC
CK0030
2018.1
Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Traversing nested lists
Some list operations
Tuples

A list of columns and a list of pairs



The first index looks up an element in the outer list

- This element can be indexed with the second index

Loops and lists
FC
CK0030
2018.1
Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Traversing nested lists
Some list operations
Tuples

Printing objects

Nested lists

Printing objects

To immediately view the nested list `table`, we may write `print table`

- Any object `obj` can be printed to screen by `print obj`

The output is usually one line, which may be very long with packed lists

Example

A long list, like the `table` variable, needs a long line when printed

```
[[-20, -4.0], [-15, 5.0], [-10, 14.0], ..., ..., [40, 104.0]]
```

Splitting the output over shorter lines makes the layout more readable

Loops and lists
FC
CK0030
2018.1
Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Traversing nested lists
Some list operations
Tuples

Printing objects (cont.)

The `pprint module` offers a **pretty print** embellishing functionality

Example

```
import pprint
pprint.pprint(table)
```

```
[[-20,  -4.0],
 [-15,   5.0],
 [-10,  14.0],
 [-5,   23.0],
 [0,    32.0],
 [5,    41.0],
 [10,   50.0],
 [15,   59.0],
 [20,   68.0],
 [25,   77.0],
 [30,   86.0],
 [35,   95.0],
 [40,  104.0]]
```

Loops and lists
FC
CK0030
2018.1
Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Traversing nested lists
Some list operations
Tuples

Printing objects (cont.)

The book offers a modified `pprint module`, named `scitools.pprint2`

- Format control over printing of `float objects` in `list objects`
- `scitools.pprint2.float_format`, as `printf format` string

Example

How the output format of real numbers can be changed

```
>>> import pprint, scitools.pprint2
>>> somelist = [15.8, [0.2, 1.7]]
>>> pprint.pprint(somelist)
    [15.800000000000001, [0.20000000000000001, 1.7]]

>>> scitools.pprint2.pprint(somelist)
    [15.8, [0.2, 1.7]]

>>> # default output is '%g', change this to
>>> scitools.pprint2.float_format = '%.2e'
>>> scitools.pprint2.pprint(somelist)
    [1.58e+01, [2.00e-01, 1.70e+00]]
```

Loops and lists
FC
CK0030
2018.1
Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Traversing nested lists
Some list operations
Tuples

Printing objects (cont.)

The `pprint module` writes floating-point numbers with lots of digits

- To explicitly facilitate detection of round-off errors

Many find this type of output annoying and prefer the default output

- `scitools.pprint2` returns a conventional output

Printing objects (cont.)

Definition

pprint and scitools.pprint2 modules have function pformat

- *It returns a formatted string, rather than printing a string*
- *It works as pprint*

```
s = pprint.pformat(somelist)
print s
```

The print statement prints like pprint.pprint(somelist)

Printing objects (cont.)

Tabular data like in `nested table lists` are not printed in a pretty way

⇝ A limitation of the `pprint module`

The expected pretty output is two aligned columns

We will have to code the formatting

⇝ To produce such output

Example

Loop over each row, extract the two elements `C` and `F` in each row

- Print these in fixed width fields
- Use the `printf` syntax

```
for C, F in table:
 print '%5d %5.1f' % (C, F)
```


Extracting sublists

Nested lists

Extracting sublists

Python has a syntax for extracting/accessing parts of a `list` structure

- **Sublists** or **slices**

Extracting sublists (cont.)

`A[i:]` refers to the sublist of `A` starting with index `i` in `A` till the end of `A`

```
>>> A = [2, 3.5, 8, 10]
    #    0    1  2   3

>>> A[2:]
    [8, 10]
```

`A[:i]` refers to the sublist of `A` starting with index of `0` in `A` till index `i-1`

```
>>> A = [2, 3.5, 8, 10]
    #    0    1  2   3

>>> A[:3]
    [2, 3.5, 8]
```

- The last index that is considered is `i-1`
- (This is important to remember)

Loops and lists
FC
CK0030
2018.1
Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Traversing nested lists
Some list operations
Tuples

Extracting sublists (cont.)

`A[i:j]` refers to the sublist of `A` starting with index `i` in `A` till index `j-1`

```
>>> A = [2, 3.5, 8, 10]
    #    0    1  2   3

>>> A[1:3]
    [3.5, 8]
```

- The last index that is considered is `j-1`
- (This is important to remember)

Loops and lists
FC
CK0030
2018.1
Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Traversing nested lists
Some list operations
Tuples

Extracting sublists (cont.)

`A[1:-1]` extracts all elements except the first and the last

- (Index `-1` refers to the last element)

```
>>> A = [2, 3.5, 8, 10]
    #    0    1  2   3

>>> A[1:-1]
 [3.5, 8]
```

`A[:]` refers to the whole list

```
>>> A[:]
 [2, 3.5, 8, 10]
```

Loops and lists
FC
CK0030
2018.1
Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Traversing nested lists
Some list operations
Tuples

Extracting sublists (cont.)

```
[[-20,  -4.0],     # table[0]
 [-15,   5.0],     # table[1]
 [-10,  14.0],     # table[2]
 [-5,   23.0],     # table[3]
 [0,    32.0],     # table[4]
 [5,    41.0],     # table[5]
 [10,   50.0],     # table[6]
 [15,   59.0],     # table[7]
 [20,   68.0],     # table[8]
 [25,   77.0],     # table[9]
 [30,   86.0],     # table[10]
 [35,   95.0],     # table[11]
 [40,  104.0]]     # table[12]
```

With nested lists, it is possible to use slices in the first index

```
>>> table[4:]
    [[0, 32.0], [5, 41.0], [10, 50.0], [15, 59.0], [20, 68.0],
    [25, 77.0], [30, 86.0], [35, 95.0], [40, 104.0]]
```

Extracting sublists (cont.)

```
[[-20,   -4.0],      # table[0]
 [-15,    5.0],      # table[1]
 [-10,   14.0],      # table[2]
 [-5,    23.0],      # table[3]
 [0,     32.0],      # table[4]
 [5,     41.0],      # table[5]
 [10,    50.0],      # table[6]
 [15,    59.0],      # table[7]
 [20,    68.0],      # table[8]
 [25,    77.0],      # table[9]
 [30,    86.0],      # table[10]
 [35,    95.0],      # table[11]
 [40,   104.0]]      # table[12]
```

We can also slice the second index, or both indices

```
>>> table[4:7][0:2]
    [[0, 32.0], [5, 41.0]]
```

`table[4:7]` makes a 3-element list

- Indices `4`, `5` and `6`

$\rightsquigarrow$ `[[0,32.0],[5,41.0],[10,50.0]]`

Slice `[0:2]` acts on it, picks its first two elements

- Indices `0` and `1`

$\rightsquigarrow$ `[[0,32.0],[5,41.0],[10,50.0]]`

Extracting sublists (cont.)

Sublists are always copies of the original list

- This is important

Example

```
>>> list_1 = [1, 4, 3]                                # Define list_1

>>> list_2 = list_1[:-1]                              # Define list_2
                                          # It is a sublist of list_1
                                          #          Elements 0 to -2
>>> list_2
    [1, 4]

>>> list_1[0] = 100                       #   First element of list_1
>>> list_1                                #         List_1 is modified
    [100, 4, 3]

>>> list_2                                #    List_2 is not modified
    [1, 4]
```

■

Extracting sublists (cont.)

Remark

Suppose that you have pre-defined/available some list

- Suppose that you extract some sublist from it
- Suppose that you modify such sublist

Whatever the modification on the sublist, the original list remains unaltered

- The *vice versa* is also true

Extracting sublists (cont.)

Remark

`B == A` is `True` if all elements in `B` equal corresponding elements in `A`

The test `B is A` is `True` if `A` and `B` are names for the same list

■

Extracting sublists (cont.)

Example

Consider the following piece of code

```
>>> A = [2, 3.5, 8, 10]
>>> B = A[:]
>>> C = A

>>> B == A
    True

>>> B is A
    False

>>> C is A
    True
```

Setting `B = A[:]` makes `B` refer to a copy of the list referred to by `A`

Setting `C = A` makes `C` refer to the same list object as `A`

Extracting sublists (cont.)

Example

Write the part of the table list of `[C, F]` rows where the degrees Celsius are between `10` and `35` (not including `35`)

```
>>> for C, F in table[Cdegrees.index(10):Cdegrees.index(35)]:
    ... print '%5.0f %5.1f' % (C, F)
    ...

 10 50.0
 15 59.0
 20 68.0
 25 77.0
 30 86.0
```

- `Cdegrees.index(10)` is the index of value `10` in the `Cdegrees` list
- `Cdegrees.index(35)` is the index of value `35` in the `Cdegrees` list

A `FOR-loop` does an equivalent job

⇝ `for C, F in table[6:11]:`

Travessing nested lists

Nested lists

Travessing nested lists

Traversing the `nested list` `table` could be done by a loop

```
for C, F in table:
 <process C and F>
```

Natural, when we know that `table` is a list of `[C, F]` lists

More general nested lists must be handled differently

- Unknown how many elements there are in each list
- (Lists are the element of the main list)

Travessing nested lists (cont.)

Example

Consider a nested list `scores` recording the scores of players in some game

- `scores[i]` holds the list of scores obtained by player number `i`

Different players have played the game a different number of times

- The length of `scores[i]` depends on `i`, the player

```
scores = []

# Hypothetical scores of player no. 0:
scores.append([12, 16, 11, 12])                          # Length 4

# Hypothetical scores of player no. 1:
scores.append([9])                                        # Length 1

# Hypothetical scores of player no. 2:
scores.append([6, 9, 11, 14, 17, 15, 14, 20])             # Length 8
```

The list has three elements, each element corresponds to a player

Loops and lists
FC
CK0030
2018.1
Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Traversing nested lists
Some list operations
Tuples

Travessing nested lists (cont.)

```
scores = []

# Hypothetical scores of player no. 0:
scores.append([12, 16, 11, 12])                          # Length 4

# Hypothetical scores of player no. 1:
scores.append([9])                                        # Length 1

# Hypothetical scores of player no. 2:
scores.append([6, 9, 11, 14, 17, 15, 14, 20])             # Length 8
```

Consider element number `g` in the list `scores[p]`, `scores[p][g]`

- It corresponds to the score in game `g` played by player `p`

The length of the individual lists `scores[p]` varies

- It equals `4`, `1`, and `8` for `p` equal `0`, `1`, and `2`, respectively

Loops and lists
FC
CK0030
2018.1
Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Traversing nested lists
Some list operations
Tuples

Travessing nested lists (cont.)

Remark

Consider `n` players, some may have played a large number of times

This makes `scores` a big nested list, potentially

Loops and lists
FC
CK0030
2018.1
Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Traversing nested lists
Some list operations
Tuples

Travessing nested lists (cont.)

Example

Consider the data initialised earlier, the table of scores

The scores can be written out in the following form

```
12 16 11 12
 9
 6  9 11 14 17 15 14 20
```

How to traverse the list and put it in table format

⇝ With well formatted columns?

The esired properties of the table formatting

1. Each row must correspond to a player
2. Columns must correspond to scores

Traversing nested lists (cont.)

```
12 16 11 12
 9
 6  9 11 14 17 15 14 20
```

We may use two nested loops

- One loop for the elements in `scores`
- One loop for the elements in the sublists of `scores`

There are two basic ways of traversing a nested list

- We use integer indices for each index
- We use variables for the list elements

Loops and lists
FC
CK0030
2018.1
Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Traversing nested lists
Some list operations
Tuples

Traversing nested lists (cont.)

An index-based version

```
scores = []
scores.append([12, 16, 11, 12])
scores.append([9])
scores.append([6, 9, 11, 14, 17, 15, 14, 20])

for p in range(len(scores)):
 for g in range(len(scores[p])):
  score = scores[p][g]
  print '%4d' % score,
 print
```

Loops and lists
FC
CK0030
2018.1
Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Traversing nested lists
Some list operations
Tuples

Traversing nested lists (cont.)

We used the trailing comma after '`print string`'

```
scores = []
scores.append([12, 16, 11, 12])
scores.append([9])
scores.append([6, 9, 11, 14, 17, 15, 14, 20])

for p in range(len(scores)):
 for g in range(len(scores[p])):
  score = scores[p][g]
  print '%4d' % score,
 print
```

The `print` after the loop over `p` adds a new (empty) line after each row

Loops and lists
FC
CK0030
2018.1
Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Traversing nested lists
Some list operations
Tuples

Traversing nested lists (cont.)

With variables for iterating over the elements in `scores` and its sublists

```
for player in scores:
 for game in player:

  print '%4d' % game,
 print
```

■

Travessing nested lists (cont.)

Definition

Consider the general case of nested lists with many indices

⇝ `somelist [i1][i2][i3] ...`

Suppose that we are interested in visiting each element in the list

We can use as many nested `FOR-loops` *as there are indices*

Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Traversing nested lists
Some list operations
Tuples

Travessing nested lists (cont.)

As a practical example consider a nested list with four indices

```
for i1 in range(len(somelist)):
 for i2 in range(len(somelist[i1])):
  for i3 in range(len(somelist[i1][i2])):
   for i4 in range(len(somelist[i1][i2][i3])):

    value = somelist[i1][i2][i3][i4]
    # perform some operation with this current value
```

This is what iterating over integer indices looks like

Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Traversing nested lists
Some list operations
Tuples

Travessing nested lists (cont.)

The corresponding version by iterating over sublists

```
for sublist1 in somelist:
 for sublist2 in sublist1:
  for sublist3 in sublist2:
   for sublist4 in sublist3:

    value = sublist4
    # perform some operation with this current value
```


Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Traversing nested lists
Some list operations
Tuples

Some list operations

Nested lists

Some list operations

Construct	Explaination
`a = []`	Initialise an empty string
`a = [1, 4.4, 'run.py']`	Initialise a list
`a.append(elem)`	Add `element`
`a + [1.3]`	Add two lists
`a.insert(i, e)`	Insert element `e` before index `i`
`a[3]`	Index a list element
`a[-1]`	Get last lists element
`a[1:3]`	Slide: Copy data to sublist
`del a[3]`	Delete an element
`a.remove(e)`	Remove an element with value `e`

Loops and lists

FC
CK0030
2018.1

Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Traversing nested lists
Some list operations
Tuples

Some list operations (cont.)

Construct	Explaination
`a.index('run.py')`	Index corresponding to element's value
`'run.py' in a`	Test if a value is in the list
`a.count(v)`	Count elements with value `v`
`len(a)`	Number of elements in list `a`
`min(a)`	The smallest element in list `a`
`max(a)`	The largest element in list `a`
`sum(a)`	Add all elements in `a`
`sorted(a)`	Return sorted version of `a`
`reversed(a)`	Return returned version of `a`
`b[3][0][2]`	Nested list indexing
`isinstance(a, list)`	`True` if `a` is a list
`type(a) is list`	`True` if `a` is a list

Loops and lists

FC
CK0030
2018.1

Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Traversing nested lists
Some list operations
Tuples

Tuples
Lists and loops

Loops and lists

FC
CK0030
2018.1

Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Traversing nested lists
Some list operations
Tuples

Tuples

Tuples are similar to `lists`, but `tuple objects` cannot be changed

- A `tuple object` can be viewed as a constant `list object`

`Lists` use square brackets, `tuples` employ standard parentheses

Tuples (cont.)

```
t = (2, 4, 6, 'temp.pdf')                                     #  Define a tuple
                                                              #          Name t

t = 2, 4, 6, 'temp.pdf'                                       #  Define a tuple
                                                              #          Name t
                                                              # W/O parenthesis
```

A comma-separated sequence of objects is a `tuple object`

- Parentheses are not necessary, though common

Loops and lists
FC
CK0030
2018.1
Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Traversing nested lists
Some list operations
Tuples

Tuples (cont.)

We can use `FOR-loop` to loop over a tuple

```
for element in 'myfile.txt', 'urfile.txt', 'herfile.txt':
 print element,
```

Note the trailing comma (`,`) in the `print` statement

```
myfile.txt yourfile.txt herfile.txt
```

The comma suppresses the final newline that `print` command would add

- The output of `print` is a `string object`

Loops and lists
FC
CK0030
2018.1
Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Traversing nested lists
Some list operations
Tuples

Tuples (cont.)

Many of the usual functionalities for lists are also available for tuples

```
>>> t = (2, 4, 6, 'temp.pdf')                                  # Define a tuple

>>> t = t + (-1.0, -2.0)                                       # Add two tuples
>>> t                                                          # Define a tuple
    (2, 4, 6, 'temp.pdf', -1.0, -2.0)

>>> t[1]                                                           # Indexing
    4

>>> t[2:]                                                      # Subtuple/slice
    (6, 'temp.pdf', -1.0, -2.0)

>>> 6 in t                                                       # Membership
    True
```

Loops and lists
FC
CK0030
2018.1
Alternative implementations
WHILE loops as FOR loops
Range construction
FOR loops with list indexes
Modify list elements
List comprehension
Multiple lists
Nested lists
Tables as row/column lists
Printing objects
Extracting sublists
Traversing nested lists
Some list operations
Tuples

Tuples (cont.)

Operations for lists that change the list do not work for tuples

```
>>> t[1] = -1
    ...
    TypeError: object does not support item assignment

>>> t.append(0)
    ...
    AttributeError: 'tuple' object has no attribute 'append'

>>> del t[1]
    ...
    TypeError: object doesn't support item deletion
```

Some methods for lists (like `index`) are not available for tuples

Tuples (cont.)

So why do we need tuples at all when lists can do more than tuples?

- ⇝ Tuples protect against accidental changes of their contents
- ⇝ Code based on tuples is faster than code based on lists
- ⇝ Tuples are often used in Python software that you will use
- (You need to know this data type!)

There is also a fourth argument, the data-type called dictionaries

- Tuples can be used as keys in dictionaries
- Lists cannot