

COMP1730/COMP6730

Programming for Scientists

Sequence types

Homework 1

- * Homework 1 will be marked in the labs this week.
- * You must attend **Your** lab group.
- * You can only change/miss your lab group in exceptional circumstances. If you let us know (by e-mail to comp1730@anu.edu.au) prior to 12:00pm today, we will try and organise an alternative. After this deadline, you must submit a request for deferred assessment through ISIS.



Lecture Outline

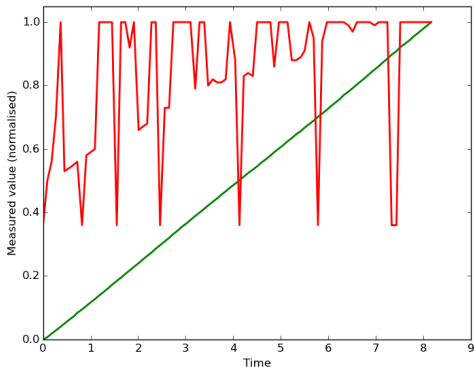
- * Sequence data types
- * Indexing & slicing
- * Sequence operations and functions
- * Iteration with `for` loops

Problem: Sensor modelling

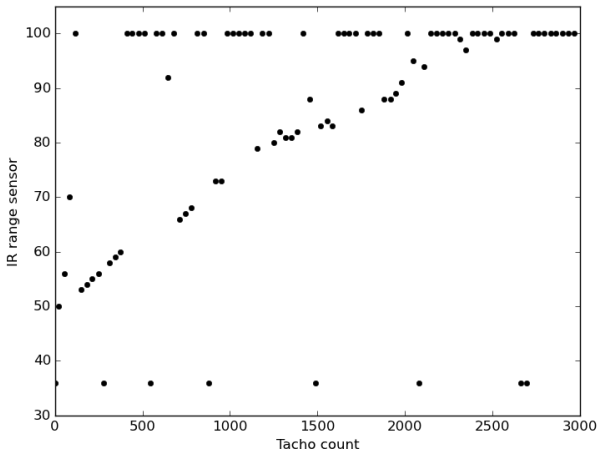
* Time series of two measurements:

* IR sensor
(% of range)

* Tachometer
(1/360th rev.)



* Is there a linear relation between x and y ?



Data Structures (Collections)

- * Provide a mechanism for grouping multiple values together.
- * There are many built in data structures in Python including `list`, `set`, `dictionary`.
- * Many more in modules such as `defaultdict` (in the `collections` module) and `heapq` (in the `heapq` module).
- * Even more in third party libraries.
- * They are optimised for different purposes.

Sequences

- * A sequence is an *ordered* collection of values.
- * Used when we want to access values based on their *position* in the sequence.
- * Each value in a sequence has a *position*, or *index*, ranging from 0 to $n - 1$.
- * The *indexing operator* can be applied to all sequence types, and returns the value at a specified position in the sequence.
 - Indexing is done by writing the index in square brackets after the sequence value, like so:

sequence [*pos*]

Sequence data types

- * python has three built-in sequence types:
 - strings (`str`) contain only text;
 - lists (`list`) can contain a mix of value types;
 - tuples (`tuple`) are like lists, but immutable.
- * Sequence types provided by other modules:
 - e.g., NumPy arrays (`numpy.ndarray`).

The list type

- * `list` is python's general sequence type.
- * To make a list, write a comma-separated list of elements in square brackets:

```
>>> x = [3.0, 1.5, 0.0, -1.5, -3.0]
>>> x
[3.0, 1.5, 0.0, -1.5, -3.0]
>>> type(x)
<class 'list'>
```

Indexing & length

list:	3.0	1.5	0.0	-1.5	-3.0
index:	0	1	2	3	4
	-5	-4	-3	-2	-1

- * In python, all sequences are indexed from 0.
- * The index must be an integer.
- * python also allows indexing from the sequence end using negative indices, starting with -1.
- * The length of a sequence is the number of elements, *not* the index of the last element.

- * Sequence elements are accessed by writing the index in square brackets, `[]`.

```
>>> x = [3.0, 1.5, 0.0, -1.5, -3.0]
```

```
>>> x[1]
```

```
1.5
```

```
>> x[-1]
```

```
-3.0
```

```
>>> len(x)
```

```
5
```

```
>>> x[5]
```

```
IndexError: list index out of bounds
```

Slicing

- * Slicing selects a subsequence of an existing sequence.

`sequence[start:end:step-size]`

- *start* is the index of the first element in the subsequence.
- *end* is the index of the first element after the end of the subsequence.
- *step-size* allows skipping of elements.
- * Slicing works on all built-in sequence types (`list`, `str`, `tuple`) and returns the same type.

Slicing Example

* More on slicing next week.

```
>>> x = [3.0, 1.5, 0.0, -1.5, -3.0]
```

```
>>> x[0:3:1]
```

```
[3.0, 1.5, 0.0]
```

```
>> x[1:5:2]
```

```
[1.5, -1.5]
```

```
>>> x[2:3:1]
```

```
[0.0]
```

```
>>> x[3]
```

```
0.0
```

Indexing vs Slicing

- * Indexing a sequence returns an element.
- * The index must be valid (i.e. between 0 and length - 1, or -1 and -length).
- * Slicing a sequence returns a subsequence of the same type.
- * A slice may contain, 0, 1 or more elements.
- * The indexes in a slice do not have to be valid.

Sequence Operations

- * The + and * operators work with sequences.
- * `sequence_1 + sequence_2` results in concatenation.

```
my_list_1 = [1, 2, 3]
my_list_2 = [2, 3, 4]
my_list_1 + my_list_2
>>> ...
```

- * `sequence * int` results in repetition.

```
my_list_1 = [1, 2, 3]
my_list_1 * 3
>>> ...
```

Functions on Sequences

- * There are many built-in functions that operate on sequences:
 - `min` and `max` return the smallest and largest elements in the sequence.
 - `sum` returns the sum of the elements in the sequence.
 - `len` returns the number of elements in the sequence.
 - `sorted` returns a `list` with the elements of the sequence arranged in ascending order.
 - `x in sequence` returns `True` iff `x` is an element of the sequence.

The `for .. in ..` statement

```
for name in expression:  
    suite
```

1. Evaluate the expression, to obtain an iterable collection.
 - If value is not iterable: **TypeError**.
2. For each element E in the collection:
 - 2.1 assign *name* the value E ;
 - 2.2 execute the loop suite.



```
my_list = [2, 3, 5, 7, 11]
for element in my_list:
    print(element * 2)
```

VS.

```
my_list = [2, 3, 5, 7, 11]
i = 0
while i < len(my_list):
    element = my_list[i]
    print(element * 2)
    i = i + 1
```

Iteration over sequences

- * Sequences are an instance of the general concept of an *iterable* data type.
 - An iterable type is defined by supporting the `iter()` function.
 - python also has data types that are iterable but not indexable (for example, sets and files).
- * The `for .. in ..` statement works on any iterable data type.
 - On sequences, the `for` loop iterates through the elements *in order*.