COMP1730/COMP6730
Programming for Scientists

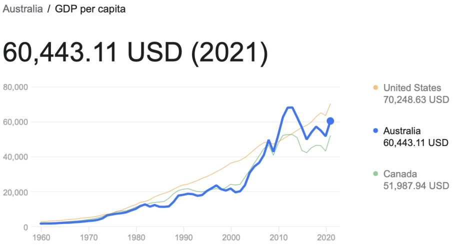
Sequence types

Lecture outline

* Sequence data types
* Indexing, length & slicing

Example sequence data

* GDP per capita (Source: Google)
* "TAACCCTAACCCTAACCC TAACCCTA..." first bit of Human genome (Source: NCBI database).



Sequences

- Sequence is an **ordered** collection of values/elements.
- A *sequence* contains zero or more values.
- Each value in a sequence has a *position*, or *index*, ranging from 0 to $n - 1$.

Sequence data types

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

Indexing & length

sequence:	`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 <u>from 0</u>.
* 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.

- `len(sequence)` returns sequence length.
- Sequence elements are accessed by writing the index in square brackets, `[]`.

```
>>> x = [3, 1.5, 0, -1.5, -3]
>>> x[1]
1.5
>> x[-1]
-3.0
>>> len(x)
5
>>> x[5]
IndexError: list index out of range
```

Functions on sequences

There are many built-in functions that operate on sequences:

* `len` returns the number of elements in the sequence.
* `min` and `max` return the smallest and largest elements in the sequence.
* `sum` returns the sum of the 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 of for
    statement1
    statement2
    ...
```

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.

Iterating over sequence elements with `for` loop

`while` loop over elements:

```
seq = [1, 4, "three", -2]
i = 0
while i < len(seq):
    print(seq[i])
    i = i+1
```

Doing the same with `for` loop:

```
seq = [1, 4, "three", -2]
for elem in seq:
    print(elem)
```

or

```
for i in range(len(seq)):
    print(seq[i])
```

Coding problem: Find the year that Australia has the highest GDP per capita.

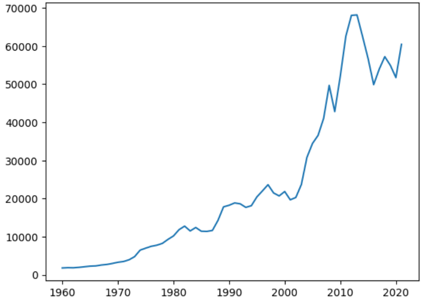
```
# GDP per capita of Australia in USD from 1960 to 2021
# Source: datacommons.org and The World Bank
gdp_au = [1811, 1878, 1855, 1967, 2131, 2281, 2344,
        2580, 2724, 2991, 3305, 3495, 3949, 4771,
        6483, 7004, 7487, 7776, 8253, 9294, 10209,
        11853, 12779, 11515, 12421, 11441, 11391,
        11651, 14284, 17834, 18250, 18860, 18625,
        17700, 18130, 20447, 22020, 23645, 21478,
        20699, 21853, 19682, 20291, 23706, 30820,
        34461, 36571, 41024, 49681, 42810, 52132,
        62599, 68047, 68156, 62515, 56709, 49877,
        53934, 57207, 54941, 51722, 60445]
```

This code was live demo during lecture:

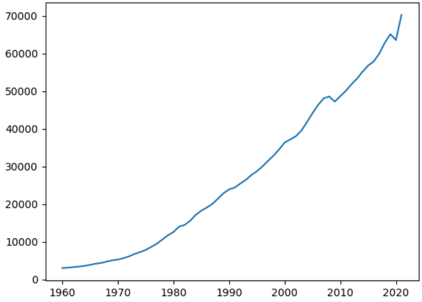
```
def find_peak(seq):
    """
        Find index of the maximum peak in a sequence
    """
    peak_id = 0
    for i in range(len(seq)):
        if seq[i] > seq[peak_id]:
            peak_id = i
    return peak_id
```

* Is there a linear relation between GDP and time?

Australia



USA



An algorithmic idea for linear regression

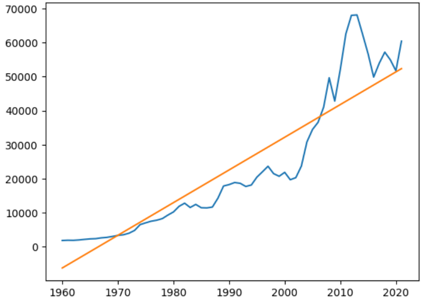
- Fit a straight line ($y = ax + b$) as close to all of the points as possible, where y is GDP and x the time.
- a is called `slope` and b is `intercept`.
  - Calculate slope as (last_y - first_y)/(last_x - first_x).
  - Calculate intercept as average over all points: $y[i] - slope * x[i]$
  - a straight line from point (first_x, slope*first_x + intercept) to point (last_x, slope*last_x + intercept)

This code was live demo during lecture:

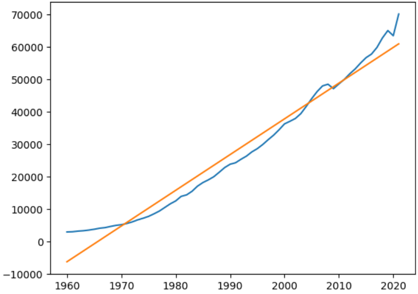
```
import matplotlib.pyplot

def linear_regression(x, y):
    """find the straight line fitting
    best to x and y,
    Assume x and y are two sequences
    of the same length"""

    slope = (y[-1] - y[0]) / (x[-1]-x[0])

    intercept = 0
    for i in range(len(y)):
        intercept = intercept + y[i] - slope*x[i]
    intercept = intercept / len(y)

    print("slope:", slope, " intercept:", intercept)
    matplotlib.pyplot.plot(x, y)
    matplotlib.pyplot.plot([x[0], x[-1]],
        [slope*x[0]+intercept, slope*x[-1]+intercept])
```

Linear regression results

Australia



USA



Generalised indexing

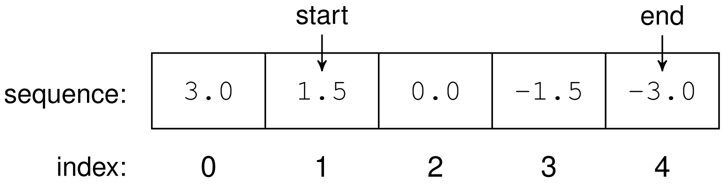
* Most python sequence types support *slicing* – accessing a subsequence by indexing a range of positions:

```
sequence[start_index:end_index]
sequence[start_index:end_index:step_size]
```

Slicing

* The slice range is "half-open": start index is included, end index is one after last included element.

```
>>> x = [3, 1.5, 0, -1.5, -3]
>>> x[1:4]
[ 1.5, 0, -1.5]
```



Slicing is an operator

* The slicing operator returns a sequence, which can be indexed (or sliced)
* What will the following print:

```
>>> x = [3, 1.5, 0, -1.5, -3]
>>> print(x[1:4][1])
```

* Slicing associates to the left.

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 returns a subsequence of the same type: Indexes in a slice do not have to be valid. And a slice may contain 0 or more elements.

Take home message

* `list` data type to store an (ordered) sequence of values.
* Sequence index starts from 0, not 1!
* Indexing operator returns an element, whereas slicing operator returns a sequence.