1811/2807/7001ICT
Programming Principles

School of Information and Communication Technology
Griffith University

Trimester 1, 2024

17 Tuples and Lists

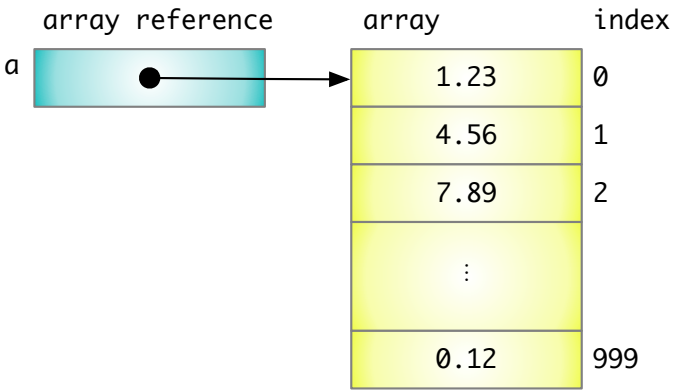
Most imperative languages use arrays for storing a sequence of values.

Python, instead, borrows tuples and lists from functional programming languages.

17.1 Arrays

In old languages like Fortran and C, and even newer ones like Java, the data type used for storing many values with just one variable name is the array.

An array is a contiguous sequence of values, all of the same type.



Definition: The *elements* of an array are the values stored in it.

Definition: Array *indices* are numbers that indicate the position within the array of the array elements.

Arrays provide very fast access to any element because its memory location can be determined by simple arithmetic.

$$\textit{element location} = \textit{array reference} + \textit{index} \times \textit{element size}$$

However old-fashioned arrays have some limitations:

- The elements must all have the same type.
- Once the array is created, its size is fixed.

Some newer languages such as JavaScript and Swift, continue to use the word *array*, but really they behave more like Python's lists, particularly in that they can grow and shrink.

17.2 Tuples

The name *tuple* is a generalisation of pair, triple, quadruple, quintuple, sextuple, septuple, ...

Tuples are a lightweight way to package several values, without defining a new `struct` as in C, or a class.

```
>>> (1, 2)
(1, 2)
>>> 1, 2
(1, 2)
>>> type((1,2))
<class 'tuple'>
>>>
```

Tuples:

- consist of multiple values separated by commas;
- maintain the values in order;
- are immutable (Once made, they can not be modified.);
- are often written enclosed in parentheses for readability, but this is not always required; and
- do not require all of the elements to be of the same type.

17.2.1 Multiple assignment

If the left hand side of an assignent is a pattern that is a tuple or list then multiple values may be assigned simultaneously.

The length of the sequence on the right must be the same as the length on the left.

Assigning two values at the same time, with one assignment statement solves the problem of swapping values in one statement.

```
b, a = a, b
```

17.2.2 Multiple return values

One of the limitations of functions (in all languages) is that functions can only return one value.

If that value is a tuple, effectively multiple values may be returned.

```
# script: tuple1.py
# A function that returns a tuple

from math import sin, cos

def polarToRect(r, theta):
   """Convert the polar coordinates (r, theta (rad)) to
      rectangular coordinates (x, y)."""
   return (r * cos(theta), r * sin(theta))
```

17.3 Lists

Lists:

- consist of multiple values separated by commas;
- maintain the values in order;
- may be empty (zero elements);
- are mutable (They *can* be modified, and their lengths may change.);
- are written out between square brackets to distinguish them from tuples; and
- do not require all of the elements to be of the same type.

17.3.1 Empty lists

An empty list can be made:

- using the `list()` constructor; or
- using the literal expression `[]`.

In the `python3` read-eval-print loop ():

```
>>> list() == []
True
>>>
```

17.3.2 Making non-empty lists

Lists may be made by:

- using the `list()` constructor with a sequence of values as its argument;

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>
```

- writing list literals, with square brackets;

```
>>> [1, True, abs]
[1, True, <built-in function abs>]
>>>
```

- by adding new elements to the end of an existing list with the `append()` method;

```
>>> xs = [1, 2, 3]
>>> xs.append(4)
>>> xs
[1, 2, 3, 4]
>>>
```

and

- with list comprehensions, a powerful idea borrowed from functional programming languages like Haskell.

```
>>> [i * i for i in range(10)]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>>
```

17.4 Operations on sequences

Tuples, lists, and strings are all sequences that can be operated upon in various ways with a common syntax.

17.4.1 Indexing

Indexing is selecting an element by its position in a sequence.

Use a single whole number in square brackets.

```
>>> t = ("Boo", 2, "you")
>>> t[0]
'Boo'
>>> xs = [4, 1, 6]
>>> xs[2]
6
>>> s = "Hello"
>>> s[1]
'e'
>>>
```

17.4.2 Negative indices

You can index from the end of a sequence, with negative indices.

The last element is at index `-1`; the second last is at `-2`; ...

```
>>> s = "Hello"
>>> s[-1]
'o'
>>> s[-2]
'l'
>>>
```

17.4.3 Slices

Slices allow the selection of a subsequence from a sequence, not just one element.

A slice looks like an array index, with square brackets, but must also have a colon between them.

On the left of the colon there may be a whole number, positive or negative, which is the index of the start of the slice (inclusive). If missing it defaults to the start of the list.

On the right of the colon there may be a whole number, positive or negative, which is the index of the end of the slice (not inclusive). If missing it defaults to the end of the list.

```
>>> s = "abcdefghijklmnopqrstuvwxyz"
>>> s[3:7]
'defg'
>>> s[:7]
'abcdefg'
>>> s[3:]
'defghijklmnopqrstuvwxyz'
>>> s[-5:]
'vwxyz'
>>> s[:]
'abcdefghijklmnopqrstuvwxyz'
>>>
```

A second colon in a slice precedes a step.

```
>>> s = "abcdefghijklmnopqrstuvwxyz"
>>> s[3:-3:2]
'dfhjlnprtv'
>>>
```

17.4.4 Catenation

The + operator joins sequences.

```
>>> (1, 2) + (3, 4)
(1, 2, 3, 4)
>>> "hi" + " " + "there"
'hi there'
>>>
```

17.4.5 Repetition

The * operator joins multiple copies of a sequence.

```
>>> "pew " * 10
'pew pew pew pew pew pew pew pew pew pew '
>>>
```

17.4.6 in/not in

The `in` and `not in` operators test whether or not a value occurs in a sequence.

For strings the "value" may be a subsequence.

The `in` and `not in` operators have the same precedence as the other relational operators.

```
>>> 2 in [1, 2, 3]
True
>>> 100 not in range(100)
True
>>> "ll" in "hello"
True
```

17.4.7 Relational operators

The sequence types may be compared with all of the relational operators.

Comparisons are performed lexicographically, and not based on length.

```
>>> [1, 2, 3] < [3]
True
>>>
```

17.4.8 Built-in functions and methods

Built-in functions that work for most sequences:

`len(s)`	length of the sequence
`max(s)`	maximum value in the sequence
`min(s)`	minimum value in the sequence

Methods that work for most sequences:

`s.index(x)`	the index at which `x` occurs in `s`
`s.count(x)`	the number of occurrences of `x` in `s`

(The `index` method has a couple of useful, optional arguments. Look them up.)

17.4.9 Modifying lists

Of tuples, strings, and lists, only lists are mutable. So these operations can only be performed on them.

(There are other mutable sequences, like `bytearray`, so some of these would also work for them.)

These operations can replace or remove part of a sequence:

`s[index] = x`	Replace one element.
`s[slice] = t`	Replace a slice with another sequence.
`del s[slice]`	Remove a slice.

These methods can modify a list:

`s.append(x)`	Adds `x` to the end of `s`.
`s.insert(index, x)`	Inserts `x` into `s` at `index`.
`s.pop(index)`	Returns and removes the element at `index` from `s`.
`s.pop()`	Returns and removes the last element of `s`.
`s.clear()`	Removes all elements from `s`.
`s.remove(x)`	Removes the first occurrence of *x* from `s`.
`s.reverse()`	Reverses the order of the elements of `s`.
`s.sort()`	Sorts the elements of list `s`.

17.4.10 Summary of operators and their precedences

highest precedence	`**`
	unary +, unary -
	`*`, `/`, `//`, `%`
	binary +, binary -
	`==`, `!=`, `<`, `>`, `<=`, `>=`, `is`, `is not`, `in`, `not in`
	`not`
	`and`
lowest precedence	`or`

17.5 Sorting and searching examples

These are classic problems of computer science and make good examples.

Most modern programming languages provide functions that perform sorting and searching, but how do computers actually do them?

> *Problem:* Write a function that sorts a list in place without using any of Python's sorting functions.
>
> *Problem:* Write a function that searches a list for the first occurence of a given value without using any of Python's searching functions.

Section summary

This section covered:

- arrays in other languages (for comparison);
- tuples, multiple assignment, multiple function return values;
- lists, list constructor, list literals;
- operations that may be performed on all the sequence types; and
- operations that may only be performed on mutable lists.