ENGG1003 - Thursday Week 2

Data types, and introduction to arrays

Steve Weller

University of Newcastle

4 March, 2021

Lecture overview

1. variables and data types §2.2
   - principles
   - live demo
2. arrays in Python §2.3
   - principles
   - live demo

1) variables and data types

- variable names – make them descriptive
- camelCase
- snake_case
- matter of preference/style/taste
  - experiment, find what works best for you

Assignment

- `x = 2`
- `x = x + 4`
- `x+= 4` is short for `x = x + 4`

The type of a variable

- types seen so far:
  - `int`
  - `float`
  - `str`
  - another (final?) type will be introduced next lecture
- explain "floating point" terminology—think of float as real number (fractional part, not an integer)
- mention "objects" only in passing

The type of a variable (ctd.)

Type function

- §2.2.4 and §2.2.5
- built-in function `type`
- type conversion
- automatic type conversion

Live demo of variables and data types

2) Arrays in Python

- simple arrays used in Monday's lecture
  - height of a ball was computed for each millisecond
  - time stored in array `t`
  - height stored in array `y`
- arrays we use in this course are imported from `numpy` library
- for each array, all array *elements* must be of the same type
  - eg: all `int`, or all `float`

Array creation and array elements

- array *index* used to identify array elements
  - Python uses *zero-based indexing*
  - indices start at zero: $0, 1, 2, \ldots$
- four common ways of creating arrays:
  - `linspace`
  - `zeros`
  - `array`
  - `copy`

#1 Linspace

- have seen `linspace` function already
- `t = np.linspace(0, 1, 1001)` creates 1001 coordinates between 0 and 1, inclusive at both ends

- `t` is the name of the array
- array indices are $0, 1, 2, \ldots$
- array elements: `t[0]`, `t[1]`, `t[2]`, ..., `t[1000]`

```
In [1]: from numpy import linspace

In [2]: x = linspace(0, 2, 3)

In [3]: x
Out[3]: array([ 0.,  1.,  2.])
```

```
In [4]: type(x)            # check type of array as a whole
Out[4]: numpy.ndarray

In [5]: type(x[0])         # check type of array element
Out[5]: numpy.float64
```

- array has 3 elements: `x[0], x[1], x[2]`
- individual array elements have type `numpy.float64`
  - `float64` is a particular float data type in NumPy
- array `x` itself has type `numpy.ndarray`

#2 Zeros function

```
In [1]: from numpy import zeros

In [2]: x = zeros(3, int)        # get array with integer zeros

In [3]: x
Out[3]: array([ 0,  0,  0])

In [4]: y = zeros(3)             # get array with floating point zeros

In [5]: y
Out[5]: array([ 0.,  0.,  0.])

In [6]: y[0] = 0.0;   y[1] = 1.0;   y[2] = 2.0     # overwrite

In [7]: y
Out[7]: array([ 0.,  1.,  2.])

In [8]: len(y)
Out[8]: 3
```

- array of `int` or array of `float`
  - but cannot mix `int` and `float` type in one array!
- `len(y)` is *length* of array `y`
- zeros(3,int)
- zeros(3)
- 0 for int, 0. for float

#3 Array function

- create an array of zeros
- pp49-50 screenshot
- Note the use of “dots” to get floating point numbers

Index out of bounds

```
In[2]: from numpy import array
In[3]: x = array([11, 12, 13])
In[4]: x
Out[4]: array([11, 12, 13])
In[5]: print(x)
[11 12 13]
In[6]: x[0]
Out[6]: 11
In[7]: x[1]
Out[7]: 12
In[8]: x[2]
Out[8]: 13
In[9]: x[3]
Traceback (most recent call last):
  File "C:\Users\srw245\Documents\teaching software\Python\venv\lib\site-packa
    exec(code_obj, self.user_global_ns, self.user_ns)
  File "<ipython-input-9-dc2cd44e899d>", line 1, in <module>
    x[3]
IndexError: index 3 is out of bounds for axis 0 with size 3

In[10]:
```

- for array with 3 elements `x[0]`, `x[1]`, `x[2]`, only legal indices are $0, 1$ and 2
- "out of bounds" error if we try and access `x[3]`

#4 Copying an array—the problem

- BE VERY CAREFUL with naive/obvious copy method
- `copy` function creates new array and copies values
  - use this method!

#4 Copying an array—how to copy

```
In [15]: from numpy import copy

In [16]: x = linspace(0, 2, 3)          # x becomes array([ 0.,  1.,  2.])

In [17]: y = copy(x)

In [18]: y
Out[18]: array([ 0.,  1.,  2.])

In [19]: y[0] = 10.0

In [20]: y
Out[20]: array([ 10.,   1.,   2.])  # ...changed

In [21]: x
Out[21]: array([ 0.,  1.,  2.])     # ...unchanged
```

- copy function— actually *creates* then copies values

Slicing an array

- needs a figure showing boxes
- ppt figure
- colon x[i:j] address all elements from index i (inclusive) to j (exclusive)
  - well that's confusing ...
- x length 6, x[0],...x[5], write x[0:5], entries 11...16
- y length 5, subset x[1:5] from y
- WARNING: When copying a slice, the same logic applies as when naively *copying* the whole array
- no screenshot needed, but illustrate live (maybe)

Slicing an array (ctd)

```
In [1]: from numpy import linspace

In [2]: x = linspace(11, 16, 6)

In [3]: x
Out[3]: array([  11.,   12.,   13.,   14.,   15.,   16.])

In [4]: y = x[1:5]

In [5]: y
Out[5]: array([ 12.,  13.,  14.,  15.])
```

Live demo of Python arrays