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 but short(-ish), not always easy!
  - ✓ waterLevel, altitudeUAV, tunnelDepth
  - ✗ w, a, t
  - ✗ averageAnnualBatteryVoltage
- camelCase
  - lower camel case eg: iPhone, macOS
  - upper camel case eg: PlayStation, YouTube
- snake_case
  - also known as lower_case_with_underscores
  - LL textbook uses this convention
- matter of preference/style/taste/context
  - experiment, find what works best for you!

Assignment

- `x = 2`
  - "variable `x` is *assigned* the value of `2`"
- `x = x + 4`
  - value of `x` is increased by `4`
  - new value of `x` over-writes old value
- abbreviations
  - `x+= 4` is short for `x = x + 4`
  - `x -= 4` $\Longleftrightarrow$ `x = x - 4`
  - `x *= 4` $\Longleftrightarrow$ `x = x * 4`
  - `x /= 4` $\Longleftrightarrow$ `x = x / 4`

Data types in Python

Python data types of variables seen so far:

- **`int`**
- **`float`**
- **`str`**
- one more data type will be introduced next week *dramatic music plays*

  - `int`—integers, eg: $0, 1, 9, -10, 452617$
  - `float`—real numbers, eg: $3.14159, -5.5, 2.0$
    - will explain "floating point number" terminology later
  - `str`—strings, eg: 'Hello ENGG1003'

Type function

- built-in function `type` returns type of a variable
  - technically: returns type of an *object*
  - we won't emphasise objects in this course
  - see elsewhere, eg: SENG1100 Object Oriented Programming

```
In [1]: x = 2

In [2]: y = 4.0

In [3]: s = 'hello'

In [4]: type(x)          # ...object named x is an integer
Out[4]: int

In [5]: type(y)          # ...object named y is a float
Out[5]: float

In [6]: type(s)          # ...object named s is a string
Out[6]: str
```

Type function (ctd.)

- *type conversion*—variables may be converted from one type to another (if it makes sense)

```
In [1]: x = 1

In [2]: y = float(x)

In [3]: y
Out[3]: 1.0
```

```
In [1]: x = 1.0

In [2]: y = int(x)

In [3]: y
Out[3]: 1
```

- *automatic type conversion*

```
In [1]: x = 2

In [2]: x = x + 4.0

In [3]: x
Out[3]: 6.0
```

Live demo of variables and data types

2) Arrays in Python

- simple arrays appeared 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
```

- can define an array of `int`, or an array of `float`
  - but cannot mix `int` and `float` type in a single array!
- `zeros(3,int)`
  - creates an array of 3 integer zeros
- `zeros(3)`
  - creates an array of 3 floating point (float) zeros
- how to tell if array of `int` or `float`?
  - look carefully ...
  - `[0, 0, 0]` for `int`
  - `[0., 0., 0.]` for `float`
- `len(y)` is *length* of array `y`

#3 Array function

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

In [2]: x = array([0, 1, 2])          # get array with integers

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

In [4]: x = array([0., 1., 2.])       # get array with real numbers
In [5]: x
Out[5]: array([ 0.,  1.,  2.])
```

- `array([0, 1, 2]` creates array of integers
- `array([0., 1., 2.]` creates array of real 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—*take care!*

```
In [10]: y = x

In [11]: y
Out[11]: array([ 0.,  1.,  2.])     # ...as expected
```

```
In [12]: y[0] = 10.0

In [13]: y
Out[13]: array([ 10.,  1.,  2.])    # ...as expected

In [14]: x
Out[14]: array([ 10.,   1.,   2.])  # ...x has changed too!
```

- changed value of an element in array `y` . . .
  . . . but it caused the same change in array `x` !
- why? Because assignment `y = x` creates another reference *to the same array that* `x` *refers to*
- be very careful with "obvious" array copy method

#4 Copying an array—doing it right

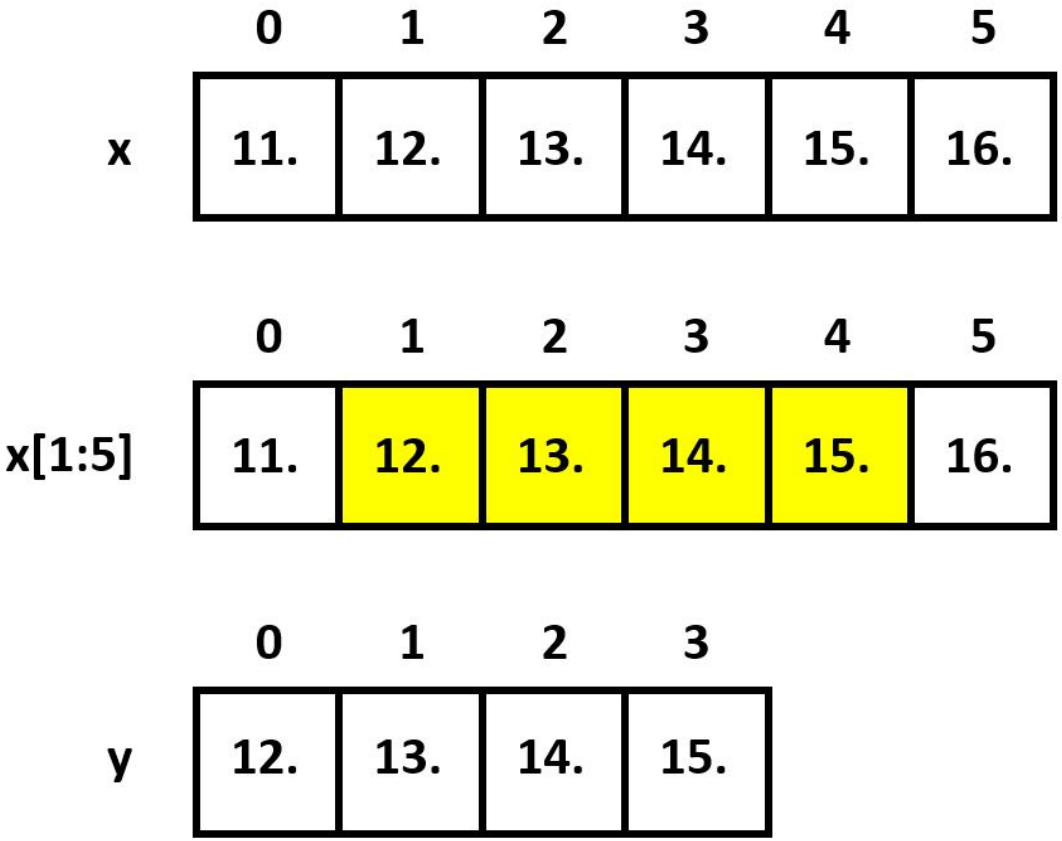
```
In [15]: from numpy import copy

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* a new array . . .

  . . . then fills with values copied from another array

Slicing an array

- *slice* of an array is a *subset* of its elements
- use colon to slice, eg: `x[i:j]`



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.])
```

`x[start:stop]` takes elements with indices `start` through `stop-1`

Example

```
x[1:5] = [x[1],x[2],x[3],x[4]
       = [12.,13.,14.,15.]
```

Live demo of Python arrays