12.010 Computational Methods of Scientific Programming

Tom Herring, tah@mit.edu

Chris Hill, cnh@mit.edu

Lecture 3: Array storage, data structures

Topics

- Continue looking at operators
- Suites and looping structures in Python.
- Logical branching – concept of decisions in code.
- Data Types
- Numbers
- Casting
- Strings
- Multi-element structures
  - Lists
  - Tuples
  - Sets
  - Dictionaries
- Arrays
- Data handling

Comparison operators

- Available: (not equal symbols vary by language e.g., MATLAB)

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Logical operators

- Allows combinations of comparison operators

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

Identity operators*

- Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location
- Class of function typically used in functions to test inputs to functions

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

Membership operators*

- Membership operators are used to test if a sequence is presented in an object.

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

Bitwise operators

- Bitwise operators are used to compare (binary) numbers

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

Boolean (logical) operations*

- Truth tables: A and B are input; OR is (inclusive) OR; XOR is exclusive OR

A	B	AND	OR	XOR
F	F	F	F	F
T	F	F	T	T
F	T	F	T	T
T	T	T	T	F

- F false is 0; T true is non-zero.
- Python has function bool that evaluates to True or False (can be a function return).
- AND and OR gates are used to add bits (AND gate generates carry bit).

Suites/Indentation

- Python uses indentation to delimit blocks of code e.g., if and for loops; function definitions.
- This approaches clearly allows the structure of the code to be seen. Good programming practice in other languages is to indent code to but it is not required.
- Other languages use keywords like “end” or “endif” to denote end if blocks (MATLAB, Fortran) or all blocks are enclosed in specific symbol e.g., C, C++ all code and blocks are enclosed in {} except for the header lines.
- Header line for Python suites end with :

Branching operations*

- If statements:
- In Python ‘if’ statement lines end in : and the True block is an indented suite.
- The ’else if’ keyword is elif:
- The else keyword is else:
- Not code validity is not checked after a logical statement is false. (This can lead to errors that occur only in some cases).

Iteration operations*

- Iteration operations are of two forms: ‘for’ and ‘while’ loops
- Loops can be broken with ‘break’ statements (allows loops to be ended before there natural end).
- Syntax
  - for x in <something>:
    - Code (indented until end of statements
  - while <logical expression>:
    - Code (indented until end of statements
  - break (with no colon) to break e.g., maybe searching for something in list and break to exit for loop when found.

Python Data types

- Python data typing happens mostly automatically but can set in code as well (referred to as casting)
- Types
  - Text Type: str
  - Numeric Types: int, float, complex
  - Sequence Types: list, tuple, range
  - Mapping Type: dict
  - Set Types: set, frozenset
  - Boolean Type: bool
  - Binary Types: bytes, bytearray, memoryview
- Determine what something is with the type function e.g., type(x)

Python Data types*

- Arrays – indexed collection of values.
  - Python 3 added Arrays as part of standard library. Prior to that arrays were in Packages.
  - Numpy package (originates in part from MIT graduate student work!) is still the primary tool for Arrays in scientific computing with Python
  - Standard library form

```
from array import *
a=array('l',[1,2,3])
print("Whole array", a)
print("Value of an element", a[0])
print("Type of array and of element",type(a),type(a[0]))
```

```
Whole array array('l', [1, 2, 3])
Value of an element 1
Type of array and of element <class 'array.array'> <class 'int'>
```

Python Data types

- Arrays – indexed collection of values.
  - Numpy form

```
import numpy as np
nel=10
a1=np.ones(nel)
a2=np.empty(nel)
a3=np.zeros(nel)
a4=np.full(nel,12,int)
a5=np.array([1,2,3.,"hello"])
```

```
print(" a1[3] = ",a1[3]) # Access element 3
print(" a2[3] = ",a2[3]) # Access element 3
print(" a3[3] = ",a3[3]) # Access element 3
print(" a4[3] = ",a4[3]) # Access element 3
print(" a5[3] = ",a5[3]) # Access element 3
```

```
 a1[3] =  1.0
 a2[3] =  1.0
 a3[3] =  0.0
 a4[3] =  12
 a5[3] =  hello
```

Python Data types*

- Arrays – indexed collection of values.
  - Numpy form

```
import numpy as np
nel=10
a1=np.ones(nel)
a2=np.empty(nel)
a3=np.zeros(nel)
a4=np.full(nel,12,int)
a5=np.array([1,2,3.,"hello"])
```

```
print(" a1[3] = ",a1[3]) # Access element 3
print(" a2[3] = ",a2[3]) # Access element 3
print(" a3[3] = ",a3[3]) # Access element 3
print(" a4[3] = ",a4[3]) # Access element 3
print(" a5[3] = ",a5[3]) # Access element 3
```

```
 a1[3] =  1.0
 a2[3] =  1.0
 a3[3] =  0.0
 a4[3] =  12
 a5[3] =  hello
```

Can have an array hold several different types. These arrays will have lower performance than simple arrays with primitive type elements (float, int etc…)

Python Data types

- Arrays – indexed collection of values.
  - Arrays using lists.
    - Initially Python did not have an array standard library. So, sometimes see Python lists used as arrays.
    - Arrays based on lists and the “array” type v numpy arrays behave differently, and we will mostly use Numpy arrays as arrays!

```
import numpy as np
a=[1,2,3]
anp=np.array([1,2,3])
```

```
print("a, anp =", a,anp)
print("a + a =", a+a)
print("anp + anp =", anp+anp)
```

```
a, anp = [1, 2, 3] [1 2 3]
a + a = [1, 2, 3, 1, 2, 3]
anp + anp = [2 4 6]
```

Python Data types*

- **Lists** are a basic type for storing indexed series of values in Python. They are a bit like arrays, but the meaning of operators is different. They are more useful for managing queues of values or things to work on than they are for mathematical/scientific formulas e.g.

```
a=[1,2,3]
a.append(4)                                          # Add a value to the end
print("Array with appended value",a)
print("Pop end value",a.pop())                       # Remove a value from the end
print("Array with appended value popped",a)
print("Index of a specific value element",a.index(3))    # Get the index of a value
print("Addition of arrays/lists", a+a)               # Adding concatenates
print(a-a)                # Other math is not defined
```

```
Array with appended value [1, 2, 3, 4]
Pop end value 4
Array with appended value popped [1, 2, 3]
Index of a specific value element 2
Addition of arrays/lists [1, 2, 3, 1, 2, 3]
TypeError: unsupported operand type(s) for -: 'list' and 'list'
```

Python Data types*

- **Dictionaries** are a way to collect keys and values. Syntax is
  - D = { “key1”: values1, “key2”: values2 }
  - Note – use of { and } and : e.g.

```
mitcourse = { 'course number': [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12] ,
              'course name': ["CEE", "Mech E", "DMSE", "Arch", "Chem", "EECS", '
```

```
nel=0
print(mitcourse["course number"][nel],mitcourse["course name"][nel])
print(mitcourse.keys())
for key, value in mitcourse.items():
    print(key, "->", value)
```

```
1 CEE
dict_keys(['course number', 'course name'])
course number -> [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
course name -> ['CEE', 'Mech E', 'DMSE', 'Arch', 'Chem', 'EECS', 'Bio', 'Phys',
'BCS', 'Chem E', 'Urban Studies', 'EAPS']
```

Python Data types

- **Dictionaries** do not have add or append operations. Instead, add a new value to a new ley (or update an existing key), e.g.

```
# To update a dictionary assig to a new (or existing) key.
mc1={'cn':7};mc1['cd']=9;mc1
```

```
{'cn': 7, 'cd': 9}
```

Python Data types*

- **Tuples** like array/list, **except** they can't be modified! Not that useful in scientific programming

```
a=(1,2,3)
```

```
print( a[0] )
a[0]=2
```

```
1
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-110-eec21a04f027> in <module>
      1 print( a[0] )
----> 2 a[0]=2

TypeError: 'tuple' object does not support item assignment
```

- you can add

```
[113]: a=(1,2,3)
       a += (7,)
       print(a)
```

```
(1, 2, 3, 7)
```

Python Data types*

- **Sets** are like mathematical sets. Only one entry for each value, no indexing, e.g

```
s1={1,2,3,3};s2={3}
print(s1.intersection(s2))
print(s1.intersection({4}))
```

```
{3}
set()
```

Summary

- Continue looking at operators
- Suites and looping structures in Python.
- Logical branching – concept of decisions in code.
- Data Types
  - Arrays
  - Lists
  - Tuples
  - Dictionary
- Next class: Numpy arrays which behave like linear algebra arrays

MIT OpenCourseWare

https://ocw.mit.edu

12.010 Computational Methods of Scientific Programming, Fall 2024