بسم الله الرّحمن الرّحيم

Outline

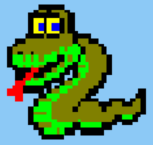
- Introduction to Python

- ABAQUS Scripting (an example)

Introduction to Python

Python

- Python is an open source scripting language.
- Developed by Guido van Rossum in the early 1990s
- Named after Monty Python
- Available for download from http://www.python.org



Outline

- Data
  - strings, variables, lists, dictionaries
- Control Flow
- Working with files
- Functions
- Classes
- Modules

Strings

- A string is a single piece of text.
  - Strings are written ’…’ or ”…”

```
>>> ”the king of spain”
the king of spain
>>> ’the king said ”hello.”’
the king said ”hello.”
```

  - Spaces are significant

```
>>> '   the   knights of   ni '
   the   knights of   ni
```

  - Backslashes mark special characters

```
>>>’hello\nworld’     # ’\n’ is a newline
hello
world
```

Operations on Strings

```
>>> ’the’ + ’king’
’theking’
>>> len(’the df’)
6
>>> ’the df’.count(’the’)
1
>>> ’the king’.replace(’the’, ’a’)
’a king’
>>> ’the king’.upper()
’THE KING’
>>> ’  hello   there   ’.strip()
’hello   there’
```

Variables

- A variable is a name for a value.
  - Use “=” to assign values to variables.

```
>>> first_name = ’John’
>>> last_name = ’Smith’
>>> first_name + ’ ’ + last_name
’John Smith’
```

  - Variable names are case sensitive
  - Variable names include only letters, numbers, and “_”
  - Variable names start with a letter or “_”
  - Any variable can hold any value (no typing)

Lists

- A list is an ordered set of values
  - Lists are written `[`*elt*$_0$, *elt*$_1$, ..., *elt*$_{n-1}$`]`

    ```
    >>> [1, 3, 8]
    [1, 3, 8]
    >>> ['the', 'king', 'of', ['spain', 'france']]
    >>> []
    >>> [1, 2, 'one', 'two']
    ```

  - **lst[*i*]** is the ***i***th element of **lst**.
  - Elements are indexed from zero

    ```
    >>> words = ['the', 'king', 'of', 'spain']
    >>> words[0]
    'the'
    >>> words[2]
    'of'
    ```

Indexing Lists

```
>>> lst = ['a', 'b', 'c', ['d', 'e']]
>>> lst[0]                          # 0th element
'a'
>>> lst[-2]                         # N-2th element
'c'
>>> lst[-1][0]                      # sub list access
'd'
>>> lst[0:2]                        # elements in [0, 2)
['a', 'b']
>>> lst[2:]                         # elements in [2, N)
['c', ['d', 'e']]
```

```
[  | 'a' ,  | 'b' ,  | 'c' ,  | ['d', 'e'] ]
   |  0     |  1     |  2     |  3
   | -4     | -3     | -2     | -1
```

Operations on Lists

```
>>> determiners = ['the', 'an', 'a']
>>> len(determiners)
3
>>> determiners + ['some', 'one']
['the', 'an', 'a', 'some', 'one']
>>> determiners
['the', 'an', 'a']
>>> determiners.index('a')
2
>>> [1, 1, 2, 1, 3, 4, 3, 6].count(1)
3
```

Operations on Lists

```
>>> determiners
['the', 'an', 'a']
>>> del determiners[2]              # remove the element at 2
>>> determiners.append('every')     # insert at the end of the list
>>> determiners.insert(1, 'one')    # insert at the given index
>>> determiners
['the', 'one', 'an', 'every']
>>> determiners.sort()              # sort alphabetically
>>> determiners
['an', 'every', 'one', 'the']
>>> determiners.reverse()           # reverse the order
['the', 'one', 'every', 'an']
```

Lists and Strings

- Converting strings to lists:

```
    >>> list('a man')          # get a list of characters
    ['a', ' ', 'm', 'a', 'n']
    >>> 'a man'.split()              # get a list of words
    ['a', 'man']
```

- Converting lists to strings:

```
    >>> str (['a', 'man'])    a representation of the list
    "['a', 'man']"
    >>> '-'.join(['a', 'man'])      # combine the list
into one string
    'a-man'
```

```
>>> i=2

>>> 'job-'+str(i)
‘job-2'
```

Dictionaries

- A dictionary maps keys to values
  - Like a list, but indexes (keys) can be anything, not just integers.
  - Dictionaries are written `{key:val, ...}`

    ```
    >>> numbers = {‘one’:1, ‘two’:2, ‘three’:3}
    ```
  - Dictionaries are indexed with `dict[key]`

    ```
    >>> numbers[‘three’]
    3
    >>> numbers[‘four’] = 4
    ```
  - Dictionaries are *unordered*.

Operations on Dictionaries

```
>>> determiners = {‘the’:‘def’, ‘an’:‘indef’,
...                ‘a’:’indef’}
>>> determiners.keys()
[‘an’, ‘a’, ‘the’]
>>> determiners.has_key(‘an’)
1                                              # 1 is true
>>> del determiners[‘an’]
>>> determiners.has_key(‘an’)
0                                              # 0 is false
>>> lastFrame.fieldOutputs['EPDDEN'].values
```

True and False

```
>>> 5 == 3+2                    # == tests for equality
True

>>> 5 != 3*2                    # != tests for inequality
False

>>> 5 > 3*2                     # >, <, >=, <= test for ordering
False

>>> 5 > 3*2 or 5<3*2     # or, and combine truth values
False
```

Control Flow

- **`if`** statement

```
if i > 3:
    del mdb.models['Model-1'].steps['load']
    print ’deleted the load step’
```

body

  – Indentation is used to mark the body.
  – Note the “:” at the end of the if line.

Control Flow

- if-elif-else statement

```
      if i == 5:
body1 {   del mdb.models['Model-1'].steps[‘preload']
      {   print ’deleted the preload step’
      elif i == 6:
body2 {   del mdb.models['Model-1'].steps['load']
      {   print ’deleted the load step’
      else:
body3 {   i
```

  - Indentation is used to mark the body.
  - Note the “:” at the end of the if line.

Control Flow

- `while` statement

body
```
while x < 1000 :
        x = x*x+3
```

  - Indentation is used to mark the body.
  - Note the “:” at the end of the if line.

Control Flow

- `for` statement

```
for n in [1, 8, 12]:
        print n*n+n

range()

for n in range(0, 10):
        print n*n
```

  - Indentation is used to mark the body.
  - Note the “:” at the end of the if line.

Working with Files

- To read a file:

```
>>> for line in open('corpus.txt', 'r').readlines()
...     print line
```

- To write to a file:

```
>>> outfile = open('output.txt', 'w')
>>> outfile.write(my_string)
>>> outfile.close()
```

- Example:

```
>>> outfile = open('output.txt', 'w')
>>> for line in open('corpus.txt', 'r').readlines()
...     outfile.write(line.replace('a', 'some'))
>>> outfile.close()
```

Functions

- A function is a reusable piece of a program.
- Functions are defined with `def`

```
>>> def square(x):
...     return x*x
>>> print square(8)
64


>>> def power(x, exp=2):        # exp defaults to 2
...     if x <= 0: return 1
...     else: return x*power(x, exp-1)
```

Classes

- A class acts as the object which contains variables and operations (or methods)
- The simplest class:

```
>>> class Simple: pass
```

- Class objects are created with the constructor, which has the same name as the class:

```
>>> obj = Simple()
```

- Variables are accessed as **`obj.var`**

```
>>> obj.x = 3
```

An Example Class

```
>>> class Account:
...       def __init__(self, initial):
...           self.balance = initial
...       def deposit(self, amt):
...           self.balance = self.balance + amt
...       def withdraw(self,amt):
...           self.balance = self.balance - amt
...       def getbalance(self):
...           return self.balance
```

- **`__init__`** defines the constructor
- **`self`** is the object that is being manipulated.
  - It is the first argument to every method.

Using the example class

```
>>> a = Account(1000.00)
>>> a.deposit(550.23)
>>> print a.getbalance()
1550.23
>>> a.deposit(100)
>>> a.withdraw(50)
>>> print a.getbalance()
1600.23
```

Modules

- A module is a collection of useful operations and objects.
- Access modules with **`import`**

```
>>> import Odb                                    # regular expressions
```

- Or use **`from…import`**

```
>>> from abaqus import *
```