Schedule

Python for Ecologists

Tom Purucker, Tao Hong, Chance Pascale

Ecological Society of America Workshop
Portland, OR

purucker.tom@gmail.com

August 3, 2012

Straw Man Schedule

- Laptop configuration 8:00-8:45
- Morning session 8:45- 10:00
  - Block 01- Introduction and Variables- Tom
  - Block 02- Strings- Tom
  - Block 03- Lists- Tom
- Mid-morning break 10:00-10:15
- Late morning session 10:15-11:45
  - Block 04- Dictionaries- Tao
  - Block 05- Functions- Tom
  - Block 06- Loops- Chance
- Lunch 11:45-1:00
- Late morning session 1:00-2:30
  - Block 07- File Input/Output- Chance
  - Block 08- Functions- Tom
  - Block 09- Object-Oriented Programming- Chance
- Afternoon break 2:30-2:45
- Late afternoon session 2:45-4:15
  - Block 10- Numpy- Tao
  - Block 11- Population Models- Tao

Introduction, Python Setup, Variables

Python for Ecologists

Tom Purucker, Tao Hong, Chance Pascale

Ecological Society of America Workshop
Portland, OR

purucker.tom@gmail.com

August 4, 2012

Notes

Python for Ecologists

- Assuming not much programming experience
- Immersion approach
  - Short lecture on Python topic
  - Hands-on Python exercises
  - Rinse & repeat
- Will use ecological examples as much as possible

Notes

Your presenters

- Tom Purucker
- Tao Hong
- Chance Pascale

Notes

Why bother with Python?

- A scripting language (like R) but also,
- A high level programming language
- Strong libraries for mathematical sciences, engineering
- Designed to produce readable code
- Cross-platform
- Open source, free
- Plays well with other technologies

Notes

übertool Python project

- http://www.ubertool.org
- Created with Python as the science engine
- Integrates easily with web technologies such as HTML, JavaScript, JQuery

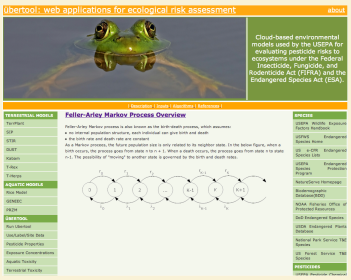


Figure: übertool ecological risk web application

Notes

Getting setup

- We will use Python 2.7 (not 3)
  - http://www.python.org/getit/
- For Windows users
  - http://portablepython.com/wiki/Download

Notes

Some extra libraries to install

- numpy- http://sourceforge.net/projects/numpy/
- scipy- http://sourceforge.net/projects/scipy/files/

Notes

Download the exercise scripts for this class

- http://www.ubertool.org
- Created with Python as the science engine

Notes

Opening a shell and running Python

- Mac- Spotlight and type 'terminal'

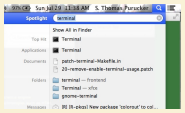


Figure: Opening terminal in OS X

- Windows- Type 'cmd' in search window for command prompt

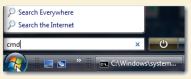


Figure: Opening the command prompt in Windows 7

Notes

Check Python installation

1. Type 'python' at the shell prompt
2. Then type at the Python prompt:

```
import sys
sys.version
import numpy
import scipy
import matplotlib
quit()
```

Notes

Run a script at the command line

```
# save this in a text file as hello.py
print "Hello Portland!"
# then navigate to its directory in a shell
# and run at the command prompt with
# python hello.py
```

Notes

Run IDLE

- IDLE is the "Interactive DeveLopment Environment" bundled with Python
- Type 'IDLE' in Mac Spotlight or Windows search window

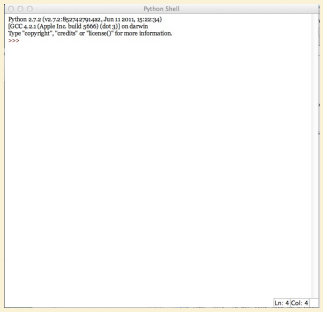


Figure: IDLE in OS X

Notes

Run hello.py with IDLE

1. Open hello.py in scripts directory with File -> Open
2. Run hello.py with Run -> Run Module or (fn) F5

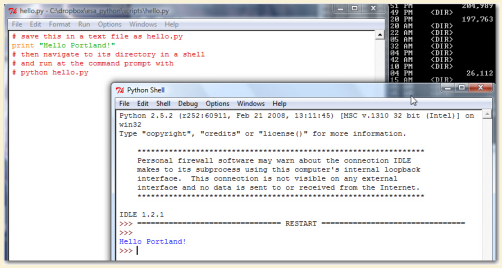


Figure: Result of running hello.py with IDLE

Notes

Variables

- No declaration of variables necessary!

```
pop_size = 112 # integer
type(pop_size)
pop_density = 4 # still an integer
type(pop_density)
pop_density = 4. # now its a float
type(pop_density)
species_name = "Oedipina_complex" # string
type(species_name)
species_name = "4" # still a string
type(species_name)
```

Notes

Basic math operations

Operation	Sign
Addition	+
Subtraction	-
Multiplication	*
Division	/
Power	**
Modulus	%

Notes

Be careful about int v float

```
>>> pop_size = 1086
>>> area = 1254
>>> pop_density = pop_size/area
>>> print(pop_density)
0
>>> type(pop_density)
<type 'int'>
```

Beware

- Declare floats by using a decimal point
- e.g., pop_size = 1086.

Notes

Python variable naming conventions

- all lowercase
- cannot start with numbers
- separate_words_with_underscores
- Style Guide for Python:
  - http://www.python.org/dev/peps/pep-0008/

Notes

unittest exercises

- Exercise 1 uses the unittest library so you can type code and test the result yourself
  1. Edit the script in IDLE between the # and the selfassert calls
  2. Run it
  3. If it complains, fix it and run it again!

Beware

- Python is very picky about space formatting, start your editing right below each # (8 spaces over)
- Python is case-sensitive- diffusion_rate and Diffusion_rate are different variables

Notes

Exercise 1- Run the script exer01_variables.py

```
import unittest

class TestVariables(unittest.TestCase):
    def test_variables(self):
        # create the variable 'diffusion_rate',
        # and assign it a float value of 6.0
        # ****************************************

        self.assertEqual(diffusion_rate, 6.)
        self.assert_(isinstance(diffusion_rate, float))

        # assign ''cohort_size'' to an integer value of 84
        # ****************************************

        self.assertEqual(cohort_size, 84)
        self.assert_(isinstance(cohort_size, int))

        # create a variable 'species_name',
        # and assign it to 'Pieza kake'
        # ****************************************

        self.assertEqual(species_name, "Pieza_kake")
        self.assertTrue(isinstance(b, str))

if __name__ == '__main__':
    unittest.main()
```

Notes

Notes

Strings

Python for Ecologists

Tom Purucker, Tao Hong, Chance Pascale

Ecological Society of America Workshop
Portland, OR

purucker.tom@gmail.com

August 4, 2012

Notes

Accessing documentation in Python

- Use dir() and help() commands to access documentation

```
import sys
dir(sys)
help(sys)
help(sys.argv)
help(str)
dir(str)
help(len)
```

- or just Google it – 'python len'

Notes

dunder methods

- "dunder" is short for double underline, e.g., __init__
- also known as special or magic methods
- + means different things for numbers and for strings

```
3 + 5
"Hello " + "world"
```

- __add__ for each type handles the different cases

Notes

len v len()

- len - refers to the object

```
help(len)
```

- len() calls the function with a supplied argument

```
len('Geospiza magnirostris')
```

Notes

Single and Double quotes

- No preference between single and double quotes

```
aint = "ain't"
aint = 'ain"t'
```

Notes

Strings

- Defining a string - single quote

```
darwins_finch = 'Geospiza magnirostris'
darwins_finch
```

- Quoting within the string - double quote

```
darwin_quote ="'Great is the power of steady misrepresentation'"
darwin_quote
darwin_quote.lower()
darwin_quote[13:18]
darwin_quote[0] #Python is zero-based
```

- Slicing strings- extracts up to, but not including the second index

```
darwin_quote[0:1]
darwin_quote[0:6]
darwin_quote[:12]
darwin_quote[12:]
darwin_quote[:12] + darwin_quote[12:]
```

Notes

Multi-line strings

- Multiline strings - triple quote

```
long_darwin_quote = '''There is grandeur in this view
of life, with its several powers, having been originally
breathed into a few forms or into one; and that, whilst
this planet has gone cycling on according to the fixed
law of gravity, from so simple a beginning endless
forms most beautiful and most wonderful have been, and
are being, evolved.'''
long_darwin_quote
print(long_darwin_quote)
len(long_darwin_quote)
```

Notes

C-like string formatting

```
>>> "%s %s" % ('Hello','Portland')
'Hello Portland'
```

Notes

Exercise 2- Run the script exer02_strings.py

```
class TestStrings(unittest.TestCase):
    def test_strings(self):
        # Create the variable ''hola'' and assign 'hello world'
        #*****************************************************

        self.assertEqual(hola, """hello world""")
        self.assert_(isinstance(hola, str))

        # Create a string, ''hola2'' that equals ''hola'' multiplied by 2
        #*****************************************************

        self.assertEqual(hola2, 'hello_worldhello_world')
.
        # Create a triple quoted string
        # ''darwin_quote2'' that has the following content:
        # Darwin said, "There is grandeur in this view of things."
        #*****************************************************

        self.assertEqual(darwin_quote2, 'Darwin_said,_"There_is_grandeur_in_this_view_of_things."')

        # Assign the method names of a string to a variable ''string_methods''
        # use ''dir()'' to list them
        #*****************************************************

        self.assertEqual(string_methods, ['__add__', ...])

        # Create a variable where_is_gra that has holds the index of the
        # substring "gra" in the string darwin_quote2.  (Find a string method to
        # figure it out)
        #*****************************************************

        self.assertEqual(where_is_gra, 23)
```

Notes

Notes

Notes

Notes

Lists

Python for Ecologists

Tom Purucker, Tao Hong, Chance Pascale

Ecological Society of America Workshop
Portland, OR

purucker.tom@gmail.com

August 4, 2012

Notes

Lists, Tuples, and Dictionaries

Type	Create Empty	Mutable?	Order
List	my_list = []	Mutable	Yes
Tuple	my_tuple = ()	Immutable	No
Dictionary	my_dictionary = {}	Mutable	No

- Mutable here means you can append, change, subtract, etc.

Notes

Lists

```
species_names = []
species_names.append("Geospiza fuliginosa") # small
species_names.append("Geospiza fortis") # medium
species_names.append("Geospiza magnirostris") # large
species_names
species_names.sort() #lists are mutable
#sorted(species_names) would not change the list
```

Notes

Lists

- Index, Value pairs
- Lists can be nested
- Tuples can use any immutable type as an index (not just integers)

Notes

Some list functions

- Lists can mix types

```
some_list = [23, 23., 'Frog', None, True]
#None and Boolean types
```

- Lists have similar methods as strings

```
some_list[0]
len(some_list)
[1,2] + [3,4]
```

- We can easily loop over the list elements

```
for thing in some_list:
  print thing
```

- And check to see if elements are in the list

```
'Frog' in some_list
'Bird' in some_list
```

Notes

Deleting list elements

- Getting rid of list elements

```
some_list
some_list.pop(0)
some_list
del some_list[2]
del some_list
```

Notes

Tuples

- Tuples are immutable objects that cannot be altered
- Use parentheses () instead of square brackets []

```
some_tuple = (23,23.,'Frog',None,True)
```

- Tuples and lists can both be sliced

```
some_tuple[0:2]
some_list[0:2]
```

Notes

Exercise 3- Run the script exer03_lists.py

```
class TestLists(unittest.TestCase):
    def test_lists(self):
        """
        A basic introduction to lists
        """
        # Create the variable ''bird_list'' and assign to an empty list
        # ************************************************
        self.assertEquals(bird_list, [])

        # Append 'American redstart' and 'Arctic tern' to ''bird_list''
        # ************************************************
        self.assertEquals(bird_list, ['American_redstart','Arctic_tern' ])

        # Sort ''bird_list''
        # ************************************************
        self.assertEquals(bird_list,  ['Arctic_tern', 'American_redstart'])

        # ''extend'' the list ''bird_list'' with ['Northern parula', 'george']
        # ************************************************
        self.assertEquals(bird_list, ['Arctic_tern', 'American_redstart', ...])

        # create a variable ''warbler_id'' with the index of 'Hooded warbler' in
        # ''bird_list'' using list methods.
        # ************************************************
        self.assertEquals(warbler_id, 3)
```

Notes

Dictionaries

Python for Ecologists

Tao Hong, Chance Pascale, Tom Purucker

Ecological Society of America Workshop
Portland, OR

purucker.tom@gmail.com

August 4, 2012

Notes

Dictionaries

- Also known as associative arrays or hashmaps
- key:value
- Are mutable, like lists
- Unlike lists, index can be something other than an integer
- Lists keep order, dictionaries don't
- Can be very efficient for searching and for table lookups

```
bw_grams = {}
bw_grams['Spring peeper'] = 4
bw_grams['Bullfrog'] = 500
bw_grams['Cane toad'] = 1800
print bw_grams['Bullfrog']
```

Notes

Setting keys and values

```
print bw_grams['Barking treefrog']
'Barking treefrog' in bw_grams
print bw_grams.get('Barking treefrog', 'Not found')
# setting a default value for a key
if 'Barking treefrog' not in bw_grams:
  bw_grams['Barking treefrog'] = 80
# another way to set a a value for key
bw_grams.setdefault('Barking treefrog', 80)
```

Notes

Mixing types

- Can be used to track properties of individuals in an individual-based model

```
male43 = {"sp":"Orca", "bw":10., "status":"suscept"}
male43["status"] = "infected"
male43
```

- Dictionary name (e.g., male43) can be nested and itself be a key

Notes

Using variables to map dictionaries

```
bw_grams = {}
frog = ''
weight = ''
```

Notes

Deleting a key

```
del bw_grams['infected']
```

Notes

Notes

Notes

Functions

Python for Ecologists

Tom Purucker, Tao Hong, Chance Pascale

Ecological Society of America Workshop
Portland, OR

purucker.tom@gmail.com

August 4, 2012

Notes

Functions

- A function takes one or more arguments and returns something
- Can return any type of data structure, including None
- Basic organization of a function:
- ```
  def some_function(arg1, arg2):
      #some statements
      return answer
  ```

Notes

whitespace and naming

- Whitespace usage is important
  - Indent consistently with 2 or 4 spaces
  - Use spaces instead of tabs, -tt
- Naming conventions
  - lower case words, cannot start with a number
  - use verbs that describe what the function does
  - underscore_between_words

```
def some_function(arg1, arg2):
    #some statements
    return answer
```

Notes

Example Function

- Argument names can be informative, helping documentation (e.g., growth_rate instead of arg1)

```
def add_two_numbers(num1, num2):
    return num1 + num2

add_two_numbers(6,5)
```

Notes
```

docstrings

- Tripe quoted comment at beginning of functions is accessible as a dunder doc (.__doc__) or help()

```
def add_two_numbers(num1, num2):
  """ this function adds two numbers
  together"""
  return num1 + num2

help(add_two_numbers)
```

Notes

Types of Arguments

- Required arguments do not have a default
- Keyword arguments can have a default value (and are optional)
- Keyword arguments are differentiated by setting equal to a value in the function argument list

```
def double_it(required1, keyword2=2):
  return required1 * keyword2

double_it(6)
double_it(6,3) #actually triples it
```

Notes

Notes

Notes

Loops

Python for Ecologists

Chance Pascale, Tom Purucker, Tao Hong

Ecological Society of America Workshop
Portland, OR

purucker.tom@gmail.com

August 4, 2012

Notes

Conditionals

- whitespace is very important, determines the close

```
if grade >= 90:
  print "A"
elif grade >= 80:
  print "B"
elif grade >= 70:
  print "C"
elif grade >= 60:
  print "D"
else:
  print "F"
```

Notes

Booleans

```
cloudy = True
rainy = False
```

Notes

Loop iteration

```
# through a list of numbers
for value in [1,2,3,4,5,6,7]:
  print value
# using range
for value in range(1,8):
 print value
```

Notes

enumeration

```
# through a list of strings with indices
watersheds = ["Suwanne", "Oconee", "Tennessee", "Flint"]
for index, value in enumerate(watersheds):
  print index, value
# can also loop through dictionaries
```

Notes

break, continue, and pass

```
# using break
for number in range(1,8):
 if number < 5:
   print number
 else:
    break
# using continue
for number in range(1,8):
 if number < 5:
   print number
   continue
# pass
for number in range(1,8):
  if number < 5:
    print number
  else:
    pass
```

Notes

Notes

Notes

File IO

Python for Ecologists

Chance Pascale, Tom Purucker, Tao Hong

Ecological Society of America Workshop
Portland, OR

chancebatwalrus@gmail.com

August 4, 2012

Notes

Overview of files

- What is a file?
- Where is a file stored?
- Reading files
- Writing files
- Opening files on the Internet
- Hairy Issues
- Problems!

Notes

What is a file?

- Technically - long strings of 0s and 1s
- Informally - a name for the place your program reads from and writes to

Notes

Where is a file stored?

- Files can be stored on your computer, or on a remote computer.
  - When files are on your computer, life is easy, ex:
    - 'myFileName.txt'
  - When files are on a remote computer, you need a specific protocol to retrieve them, ex:
    - 'http://mywebsite.com/myFileName.txt'

Notes

How do I get data out of a file?

1. Open the file.

```
file = open("myFileName.txt")
```

2. Loop through the File (here we loop through)

```
for line in file:
    print(line)    # line is a string, lets print it!
```

3. Close the File

```
file.close()
```

Notes

I want to do more than print!

- Code:

```
file = open('myFileName.txt')
for line in file:
  print(line)    # line is a string, lets print it!
file.close()
```

- line is a string, so we can do whatever we want to it. If the line is a sentence, and we wanted to break it up by words, we could do this instead :

```
file = open('myFileName.txt')
for line in file:
    for word in line.split():
        print(word)    # lets print each word!
file.close()
```

Notes

I want to save a file

- We can open a file for writing, as well as reading.

```
file = open('anotherFile.txt', 'w')
file.write('This is output!\n')
file.close()
```

- Be careful not to overwrite an important file!

Notes

How can we open a file from the Internet?

- Short answer: do different things
- Long answer: make one function so that both operations look the same:
  - The 'openAnything' function tries to open both URLs and local drives

```
import fileUtils
file = fileUtils.openAnything('YourFilePathHere')
```

Notes

Why you'll hate files

- Files can be open in different modes
  - w - write
  - a - appends
  - r - read (default)
  - and more!
- Files involve string parsing and manipulation
  - This can be a real pain!
  - Corruption prone!

Notes

Exercise

1. Read a file
   1. Read StateoftheUnion.txt
   2. Output each word on its own line
2. Sum a file
   1. Read IntegerFile.txt, which has many numbers on one line.
   2. Calculate the sum, and the mean. Hint, the int() function turns a string into a number.
3. File copier
   1. Copy http://www.ubertool.org/index.html to a file on your local disk.

Notes

Notes

Notes

Classes

Python for Ecologists

Tom Purucker, Tao Hong, Chance Pascale

Ecological Society of America Workshop
Portland, OR

purucker.tom@gmail.com

August 4, 2012

Notes

Python objects

- Everything in Python is an object with these properties
  1. an identity (id)
  2. a type (type)
  3. a value (mutable or immutable)

Notes

Each Python object has an id

```
>>> n_predators = 12
>>> id(n_predators)
4298191056
```

Notes

Each Python object has a type

```
>>> n_predators = 12
>>> type(n_predators)
<type 'int'>
```

Notes

Each Python object has a value

- String, integer, and tuple object values are *immutable*

```
>>> n_prey = 88
>>> id(n_prey)
4298193184
>>> n_prey = 96
>>> id(n_prey)
4298192992 # id for n_prey has changed
```

- Dictionary and list items are *mutable*

```
>>> birds = ["cardinal", "oriole"]
>>> id(birds)
4332756000
>>> birds.append("gnatcatcher")
>>> id(birds)
4332756000 # id is still the same
```

Notes

Classes

- Classes consist of
  - collections of data structures
  - collections of methods (functions)
- Class methods typically operate on the data structures of the class
- Class users then call methods and do not have to manipulate the data

Notes

self variable

- A class instance refers to itself as 'self'
- All methods require self as the first argument/parameter inside the class
- But users of the class do not include it in calls to the methods
- All data and methods calls are preceded by self within the class (e.g., self.age() or self.find_integral(some arguments...)

Notes

Creating a class

- object is the base class
- dunder init is a constructor
- all methods take self as the first argument/parameter

Notes

Code for creating a class

```
#create the Rabbit class, starts with 10 hit points
class Rabbit(object):
  def __init__(self, name):
    self.name = name
    self.hit_points = 10

  def hop(self):
    self.hit_points = self.hit_points - 1
    print "%s hops one node, now has %i hit points."
    % (self.name, self.hit_points)

  def eat_carrot(self):
    self.hit_points = self.hit_points + 3
    print "%s munches a carrot, now has %i hit points."
    % (self.name, self.hit_points)
```

Notes

Code to create some rabbits

- We can now create objects of Rabbit class and give them names

```
#create some Rabbits
were = Rabbit("Were-Rabbit")
harvey = Rabbit("Harvey Rabbit")
jessica = Rabbit("Jessica Rabbit")
dir(jessica)
```

Notes

Code to create some rabbits

- We can now create objects of Rabbit class and give them names

```
#create some Rabbits
were = Rabbit("Were-Rabbit")
harvey = Rabbit("Harvey Rabbit")
jessica = Rabbit("Jessica Rabbit")
dir(jessica)
```

Notes

Call the methods of the created rabbits

- We can now create objects of Rabbit class and give them names

```
#Rabbits hop around and eat carrots
were.hop()
jessica.eat_carrot()
harvey.hop()
jessica.hop()
were.eat_carrot()
```

Notes

Create a frog subclass

- Subclasses can inherit the data and methods of the original class and extend them

```
#Create a Frog class that extends the rabbit class
class Frog(Rabbit):
    # create a new croak method
    def croak(self):
        self.hit_points = self.hit_points - 1
        print "%s croaks, now has %i hit points."
        % (self.name, self.hit_points)
  # override the eat_carrot method
    def eat_carrot(self):
        print "%s cannot eat a carrot, it is too big!."
        % (self.name)
  # create an eat_fly method
    def eat_fly(self):
        self.hit_points = self.hit_points + 2
        print "%s eats a fly , now has %i hit points."
        % (self.name, self.hit_points)
```

Notes

Create Frog objects and call its methods

```
## Create a frog
frogger = Frog("Frogger")
# Do frog stuff
frogger.croak()
frogger.eat_carrot()
frogger.eat_fly()
frogger.hop()
```

Notes

Notes

Notes

Object-Oriented Programming

Python for Ecologists

Chance Pascale, Tom Purucker, Tao Hong

Ecological Society of America Workshop
Portland, OR

chancebatwalrus@gmail.com

August 4, 2012

Notes

Principles of OOP

- Classes
  - Inheritance
  - Abstraction
  - Encapsulation
  - Polymorphism
- Methods
- Decoupling

Notes

Let's take a gamble on classes

- Fundamental data unit for card games is Card
- Collection of Cards is Deck
- Subset of the Deck is Hand
- You need a CardGame to do something with the Cards, Deck, and Hands
- OldMaidGame is an type of CardGame
- OldMaidHand is a type of Hand used in OldMaidGame

Notes

ULM a nice city in Germany, oops UML

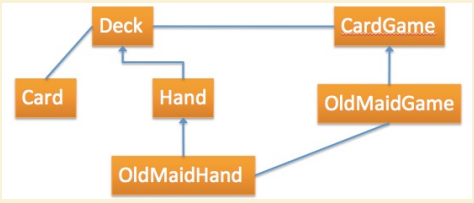


Figure: UML of OldMaidGame

Notes

Card Class Design

- Each card has a suit and a value
- Suits have no intrinsic use outside of a card so no real need to create a class for them
- Values are sometimes integers and other times strings, so represent them as string and associate true value for each game

Notes

Card Class Code

```
class Card:
  suitList=["Clubs", "Diamonds", "Hearts", "Spades"]
  rankList=['Ace','2','3','4','5','6','7','8','9','10','Jack','Queen','King']

  def __init__(self, suit = 0, rank = 2):
    self.suit = suit
    self.rank = rank

  def __str__(self):
    return (self.rankList[self.rank] + "␣of␣" + self.suitList[self.suit])

  def __cmp__(self, other):
if self.suit > other.suit: return 1
    if self.suit < other.suit: return -1
if self.rank > other.rank: return 1
    if self.rank < other.rank: return -1
return 0
```

Notes

Deck class design

- A Deck is a collection of cards
- General functionality of decks are that they can be shuffled, the 'top' card can be drawn, and many times knowing if there are any cards in the deck is necessary

Notes

Deck Class Code

```
class Deck:
  def __init__(self):
    self.cards = []
    for suit in range(4):
      for rank in range(1, 14):
        self.cards.append(Card(suit, rank))

  def printDeck(self):
    for card in self.cards:
      print card

  def __str__(self):
    s = ""
    for i in range(len(self.cards)):
      s = s + "␣" * i + str(self.cards[i]) + "\n"
    return s
```

Notes

Deck Class Code (continued)

```
  def shuffle(self):
    import random
    nCards = len(self.cards)
    for i in range(nCards):
      j = random.randrange(i, nCards)
      self.cards[i],self.cards[j]=
          self.cards[j],self.cards[i]
  def removeCard(self, card):
    if card in self.cards:
      self.cards.remove(card)
      return True
    else:
      return False
  def popCard(self):
    return self.cards.pop()
  def isEmpty(self):
    return (len(self.cards) == 0)
```

Notes

Hand Class Design

- Hand is an example of a Deck, a subset of the complete deck
- If we make Hand inherit from Deck, then we get the data structures that Deck contains
- Hand can also call the methods of Deck as if it was the superclass
- If a method of Deck class is not included in Hand code, then if that method is called on Hand object Deck method will be used.

Notes

Hand Class Code

```
class Hand(Deck):
  pass
  def __init__(self, name = ""):
    self.cards = []
    self.name = name

  def addCard(self, card):
    self.cards.append(card)

  def deal(self, hands, nCards = 999):
    nHands = len(hands)
      for i in range(nCards):
        if self.isEmpty():
          break # break if out of cards
        card = self.popCard() # take the top card
        hand = hands[i % nHands] # whose turn is next?
        hand.addCard(card) # add the card to the hand
```

Notes

Hand Class Code (continued)

```
# if next function not in code
# Deck __str__ would be called
  def __str__(self):
    s = "Hand_" + self.name
    if self.isEmpty():
      return s + "_is_empty\n"
    else:
      return s + "_contains\n" + Deck.__str__(self)
```

Notes

CardGame Class Design/Code

- Card games contain a single deck, which should be shuffled at the beginning of each game

```
class CardGame:

  def __init__(self):
    self.deck = Deck()
    self.deck.shuffle()
```

Notes

OldMaidHand Design/Code

- Pretty much a normal card hand but needs method to find and remove all matches and return a count of matches

```
class OldMaidHand(Hand):
  def removeMatches(self):
    count = 0
    originalCards = self.cards[:]
    for card in originalCards:
      match = Card(3 - card.suit, card.rank)
      if match in self.cards:
        self.cards.remove(card)
        self.cards.remove(match)
        print "Hand %s: %s matches %s" %
              (self.name, card, match)
        count = count + 1
    return count
```

Notes

OldMaidGame Design

- Like all classes that "extend" CardGame, a deck is needed and it should be shuffled, oh wait CardGame already does this
- Playing the game performs the following steps:
  1. Take the Queen of hearts out of the deck
  2. Deal OldMaidHands
  3. Remove and count number of matches
  4. Each turn for a player is the same(25 turns):
     1. Check if hand is empty, if so do nothing
     2. Take neighbor playerâĂŹs âĂIJtopâĂİ card
     3. Remove and count number of matches
     4. Shuffle your hand
  5. Top score after all turns is winner

Notes

OldMaidGame Code

```
class OldMaidGame(CardGame):
    def play(self, names):
        self.deck.removeCard(Card(0, 12)) # remove Queen of Clubs
        self.hands = [] # make a hand for each player
        for name in names :
            self.hands.append(OldMaidHand(name))
            # deal the cards
            self.deck.deal(self.hands)
            print "---------- Cards have been dealt"
            self.printHands()
            # remove initial matches
            matches = self.removeAllMatches()
            print "---------- Matches discarded, play begins"
            self.printHands()
    turn = 0  # play until all 50 cards are matched
        numHands = len(self.hands)
        while matches < 25:
            matches = matches + self.playOneTurn(turn)
            turn = (turn + 1) % numHands
            print "---------- Game is Over"
            self.printHands()
```

Notes

OldMaidGame Code (continued)

```
def removeAllMatches(self):
        count = 0
        for hand in self.hands:
            count = count + hand.removeMatches()
        return count

def playOneTurn(self, i):
        if self.hands[i].isEmpty():
            return 0
        neighbor = self.findNeighbor(i)
        pickedCard = self.hands[neighbor].popCard()
        self.hands[i].addCard(pickedCard)
        print "Hand", self.hands[i].name, "picked", pickedCard
        count = self.hands[i].removeMatches()
        self.hands[i].shuffle()
        return count
```

Notes

OldMaidGame Code (continued)

```
def findNeighbor(self, i):
        numHands = len(self.hands)
        for next in range(1, numHands):
            neighbor = (i + next) % numHands
            if not self.hands[neighbor].isEmpty():
                return neighbor
```

Notes

What is __init__.py

- Files named __init__.py are used to mark directories on disk as a Python package directories. If you have the files
- mydir/spam/__init__.py mydir/spam/module.py and mydir is on your path, you can import the code in module.py as:
- import spam.module or
- from spam import module If you remove the __init__.py file, Python will no longer look for submodules inside that directory, so attempts to import the module will fail.
- The __init__.py file is usually empty, but can be used to export selected portions of the package under more convenient names, hold convenience functions, etc. Given the example above, the contents of the __init__ module can be accessed as
- import spam

Notes

Notes

Numpy
Python for Ecologists

Tao Hong, Tom Purucker , Chance Pascale

Ecological Society of America Workshop
Portland, OR
hongtao510@gmail.com
August 1st, 2012

1

Overview of files

- Install Numpy
- Array
- Indexing
- Matrix
- Reference

2

Install Numpy

- Windows
  - 1. for 32 bit machine, download from: http://sourceforge.net/projects/numpy/files/NumPy/1.6.2/
  - 2. for 64 bit system, download from: http://www.lfd.uci.edu/~gohlke/pythonlibs/
  - **Portable Python** http://www.portablepython.com/wiki/PortablePython2.7.3.1

3

Create an array

create an array from a list

>>>import numpy as np

>>>a=np.array([10, 20, 30, 40])

>>>a

>>>> [10 20 30 40]

>>> a.shape

>>>(4,)

4

np.array (cont'd)

>>> b = np. array([[1, 2, 3, 4],[4, 5, 6, 7], [7, 8, 9, 10]])
>>> b
>>> [[1 2 3 4]
[4 5 6 7]
[7 8 9 10]]
>>> b.shape
>>>(2,2)
>>> b.reshape(4,3)
>>> [[1 2 3]
[4 4 5]
[6 7 7]
[8 9 10]]

5

np.array (cont'd)

>>> b.shape = 6,-1
Python will automatically calculate the length of second axis 12/6=2
>>>b
>>> [[1 2]
[3 4]
[4 5]
[6 7]
[7 8]
[9 10]]

6

np.array (cont'd)

>>>b = np.array([[1, 2, 3, 4],[4, 5, 6, 7], [7, 8, 9, 10]],dtype=float)

>>>b.dtype

>>> float64

>>>b

>>> [[1. 2. 3. 4.]
[4. 5. 6. 7.]
[7. 8. 9. 10.]]

7

np.arange

np.array approach is not efficient, let's try np.arange

np.arange(*start*, *stop*, *step*, *dtype=None*)

>>>np.arange(0,4,1) #Is '4' included?

>>>[0 1 2 3] (does not include '4')

>>> np.arange(4)

>>>[0 1 2 3]

8

np.linespace

np.linspace(start, stop, num, endpoint=True, retstep=False)

>>>a = np.linspace(2.0, 3.0, num=5, endpoint= True, retstep=True)

>>>(array([2. , 2.25, 2.5 , 2.75, 3.]), 0.25)

>>>b = np.linspace(2.0, 3.0, num=5, endpoint= False, retstep=True)

>>>(array([2. , 2.2, 2.4, 2.6, 2.8]), 0.2)

9

Structured (record) Arrays

- allows access to its data using named fields.

>>>persontype = np.dtype({'names':['name', 'age', 'weight'],'formats':['S32','i', 'f']})

>>>a = np.array([('Name A',32,75.5),('Name B',24,65.5)], dtype=persontype)

>>>a

>>>[('Name A', 32, 75.5) ('Name B', 24, 65.5)]

10

>>a[0]

>>>('Name A', 32, 75.5)

>>>a['name']

>>>['Name A' 'Name B']

>>>a['age'][0]

>>>32

>> a['name'][1]='tao' #modify

>>>a

>>> [('Name A', 32, 75.5) ('tao', 24, 65.5)]

11

fromfunction

- Construct an array by executing a function over each coordinate

>>>def func(i):

>>> return i%4+1

>>> np.fromfunction(func, (5,))

>>> [1. 2. 3. 4. 1.]

12

Some propertites

```
>>>a = np.array([[1, 2, 3, 4],[4, 5, 6, 7], [7, 8, 9, 10]],dtype=float)
>>>[[ 0 10 20]
     [30 40 50]]

>>>a.shape          #shape
>>>(2, 3)

>>>a.ndim           #number of dimensions
>>>2

>>>a.dtype          #data type
>>> int32

>>>a.size           #number of elements
>>>6

>>> k= a.flat       #return a flat iterator over an array
>>>for i in k:
 >>>  print i
```

13

Indexing

>>> a = np.arange(10)

Integer index

>>> a[5]

0	1	2	3	4	**5**	6	7	8	9

>>5

A range (starts at the 3th and ends before 5th)

>>> a[3:5]

0	1	2	**3**	**4**	5	6	7	8	9

>>>[3, 4]

14

The first five elements

>>> a[:5]

0	**1**	**2**	**3**	**4**	5	6	7	8	9

>>>[0, 1, 2, 3, 4]

Counting backwards

>>> a[:-1]

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	9

array([0, 1, 2, 3, 4, 5, 6, 7, 8])

15

Reverse the array

>>> a[::-1]

9	8	7	6	5	4	3	2	1	0

>>>[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

```
>>> a[::-1][::2]      #every other one
>>>[ 9, 7, 5, 3, 1]

Two dimensions
>>b = np.array([[1, 2, 3, 4],[4, 5, 6, 7], [7, 8, 9, 10]])
>>c= b[:,::-1]
b= [[ 1  2  3  4]                c= [[ 4  3  2  1]
    [ 4  5  6  7]                    [ 7  6  5  4]
    [ 7  8  9 10]]                   [10  9  8  7]]
```

16

Modify content

>>> a[2:4] = 100,101

>>> a

0	1	**100**	**101**	4	5	6	7	8	9

>>> [0, 1, 100, 101, 4, 5, 6, 7, 8, 9]

a[i:j:k] *i* is the starting index, *j* is the stopping index, and *k* is the step.

>>> a[1:-1:2]

0	**1**	100	**101**	4	**5**	6	**7**	8	9

>>>[1, 101, 5, 7]

17

```
>>>x = np.arange(10)
>>> [0 1 2 3 4 5 6 7 8 9]
>>>x[[3, 3, -3, 8]]   #list
>> [3 3 7 8]
```

0	1	2	**3**	4	5	6	**7**	**8**	9

```
>> x[np.array([3,3,-3,8])] #array
>> [3 3 7 8]

>> x[np.array([True, False, True, False, False])]
>> [0 2]        #the missing ones considered as 'False'
```

18

Multidimensions

a = np.arange(0, 60, 10).reshape(-1, 1)+np.arange(0, 6)

0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

19

```
>>>a[:,2]
>>> [ 2 12 22 32 42 52]

>>>a[0,3:5]
>>>[3,4]

>>>a[4:,4:]  #select a 'block'
>>> [[44 45]
     [54 55]]
```


```
>>>a[2::2,::2]      #start from 3rd row, step_row=2, start from 1st col, step_col=2
>>>[[20 22 24]
     [40 42 44]]

>>>x[2::2,::-1]?
>>> [[25 24 23 22 21 20]
     [45 44 43 42 41 40]]
```

20

np.indices

- Return an array representing the indices of a grid
  x = np.arange(20).reshape(5, 4)
  x=[[0 1 2 3]
  [4 5 6 7]
  [8 9 10 11]
  [12 13 14 15]
  [16 17 18 19]]
  row, col = np.indices((2, 3))
  x[row, col]=[[0 1 2]
  [4 5 6]]
- Extract the required elements directly with ``x[:2, :3]``.

21

Broadcasting

- Deal with inputs that do not have exactly the same shape.
- Rule 1: if arrays do not have the same number of dimensions, then a "1" will be repeatedly added to the shapes of the smaller arrays
- Rule 2: arrays with a size of 1 along a particular dimension act as if they had the size of the array with the largest shape along that dimension.

22

```
>>> a = np.arange(0, 60, 10).reshape(-1, 1)    # a.shape=(6,1)
>>> b = np.arange(0, 5)                        #b.shape=(5,)
>>> c = a+b
How does array c looks like?

>>> a
>>>[[ 0]
     [10]
     [20]
     [30]
     [40]
     [50]]
```

Rule 1

```
>>> a = a.repeat(5, axis=1)
        >>>a
>>> [[ 0  0  0  0  0]
     [10 10 10 10 10]
     [20 20 20 20 20]
     [30 30 30 30 30]
     [40 40 40 40 40]
     [50 50 50 50 50]]
```

23

```
>>>b = np.arange(0, 5)
>>>b
>>> [0, 1, 2, 3, 4]
>>>b.shape
>>>(5,)
```

Rule 1

```
>>> b.shape=1,5
>>> b
>>>[[0, 1, 2, 3, 4]]
```

Rule 2

```
>>> b = b.repeat(6,axis=0)
>>> b
>>> [[0, 1, 2, 3, 4],
     [0, 1, 2, 3, 4],
     [0, 1, 2, 3, 4],
     [0, 1, 2, 3, 4],
     [0, 1, 2, 3, 4],
     [0, 1, 2, 3, 4]]
```

24

```
>>> c = a + b
>>> c
>>> [[ 0  1  2  3  4]
     [10 11 12 13 14]
     [20 21 22 23 24]
     [30 31 32 33 34]
     [40 41 42 43 44]
     [50 51 52 53 54]]
```

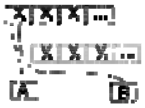
25

Deep copy shallow copy

- A shallow copies collection structure, not the elements. With a shallow copy, two collections now share the individual elements.

Shallow copy

```
>>>a = np.arange(0, 60, 10)
>>>b=a
>>>a
>>>[ 0 10 20 30 40 50]
>>>b
>>>[ 0 10 20 30 40 50]
>>>b[0]=100
>>>b
>>>[100  10  20  30  40  50]
>>>a
>>>[100  10  20  30  40  50]
```

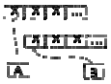


26

- Deep copies duplicate everything. A deep copy of a collection is two collections with all of the elements in the original collection duplicated.

Deep copy

```
>>>a = np.arange(0, 60, 10)
>>> b= copy.deepcopy(a)
>>>a
>>>b[0]=100
>>>b
>>>[100  10  20  30  40  50]
>>>a
>>>[0  10  20  30  40  50]
```



27

np.array VS np.mat

```
>>> a=np.mat('4 3; 2 1')
>>>a
>>> [[4 3]
     [2 1]]
>>> b=np.mat('1 2; 3 4')
>>>b
>>> [[1 2]
     [3 4]]
>>>c=a*b
>>>c
>>> [[13 20]
     [ 5  8]]
```

```
>>> a= np.array([[4, 3], [2, 1]])
>>>a
>>> [[4 3]
     [2 1]]
>>> b= np.array([[1, 2], [3, 4]])
>>>b
>>> [[1 2]
     [3 4]]
>>>c=a*b
>>>c
>>> [[4 6]
     [6 4]]
>>>d=np.dot(a,b)
>>> [[13 20]
     [ 5  8]]
```

28

```
>>>A = matrix( [[1,2,3],[11,12,13],[21,22,23]]) # Creates a matrix.
>>> [[ 1  2  3]
     [11 12 13]
     [21 22 23]]

>>>print A.T            # Transpose of A
>>> [[ 1 11 21]
     [ 2 12 22]
     [ 3 13 23]]

>>>print A.I            # Inverse of A.
>>> [[  3.00239975e+14  -6.00479950e+14   3.00239975e+14]
     [ -6.00479950e+14   1.20095990e+15  -6.00479950e+14]
     [  3.00239975e+14  -6.00479950e+14   3.00239975e+14]]
```

29

Solve a linear system

$$[a]_{3\times3}[x]_{3\times1} = [b]_{3\times1}$$

```
>>> a= np.random.rand(3,3)
>>> [[ 0.26835516  0.1812329   0.07554446]
     [ 0.2915491   0.27213494  0.05657924]
     [ 0.89496488  0.35577792  0.88181086]]
>>> b= np.random.rand(3).reshape(3,-1)
>>> [[ 0.80181235]
 [ 0.13312618]
 [ 0.5599297 ]]
>>>x=np.linalg.solve(a,b)
>>> [[ 0.83270995]
     [ 0.43698752]
     [-0.144376 ]]
```

30

More numpy matrix functions

- **NumPy for MATLAB Users**
http://www.scipy.org/NumPy_for_Matlab_Users

- **NumPy for R (and S-Plus) users**
http://mathesaurus.sourceforge.net/r-numpy.html

31

Reference

- Official document
NumPy User Guide (not the reference guide)
http://docs.scipy.org/doc/numpy/numpy-user.pdf
- Guide to NumPy by Travis E. Oliphant
http://www.tramy.us/numpybook.pdf
- http://stackoverflow.com/

32

Generic Python for Ecologists

Tao Hong, Tom Purucker, Chance Pascale

Ecological Society of America Workshop
Portland, OR
hongtao510@gmail.com
August 1st, 2012

Population models in Übertool

1. Exponential
2. Logistic
3. Gompertz
4. Fox Surplus yield
5. Maximum Sustainable Yield
6. Yule-Furry
7. Feller-Arley
8. Leslie

Exponential Model

- Mathematical equation

$$N_t = N_0 e^{rt}$$

  - N_0 is the initial number of individuals
  - N_t is population size at t
  - r is intrinsic growth rate
  - t is duration
- We have three inputs and one output

Code in Python (exp)

- Define a function

```
1 def exponentialgrow(N_o , r,  T):
2     index_set = np.arange(T+1)      #How to do this in np.linspace?
                                      #index_set = np.linspace(0,T,T+1)
3     x = np.zeros(len(index_set))    #create an array to hold the results
4     x[0] = N_o                      #initial condition
5     for t in index_set[1:]:         #t starts at 0, ends at T
6         x[t] = N_o*np.exp(r*t)
7     return x
```

- Call the defined function

```
>>>N_t=exponentialgrow(10, 0.4, 10)
>>>[ 10.         14.91824698  22.25540928  33.20116923  49.53032424
   73.89056099 110.23176381 164.44646771 245.32530197 365.98234444
   545.98150033]
```

It is your turn now!

- Please code the logistic population model

$$N_t = N_{t-1} + r_0 N_{t-1}(1 - \frac{N_{t-1}}{K})$$

  - N_t is population size at t
  - N_{t-1} is population size at t-1
  - K is population capacity
  - r_0 is max growth rate
  - t is simulation duration
- We have four inputs and one output

Code in Python (logistic)

- Define a function

```
1. def logisticgrow(N_o, T, r, K):
2.     index_set = np.arange(T+1)
3.     x = np.zeros(len(index_set))
4. x[0] = N_o
5.     for t in index_set[1:]:
6.         x[t] = x[t-1] + (r)*x[t-1]*(1 - x[t-1]/float(K))
7. return x
```

Why use float(K)?
Try 10/100
And 10/float(100)

- Call the defined function

```
>>>N_t=logisticgrow(10, 10, 0.4, 100)
>>>[ 10.         13.6        18.30016    24.28064058 31.63469878
 40.28556162 49.90808037 59.90804657 69.51536903 77.99197051
 84.85776886]
```

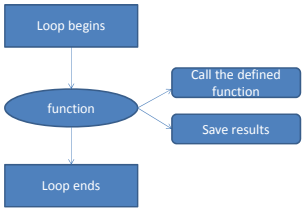
Feller-Arley (birth-death) Markov Process

- No internal population structure and each individual can give birth and death with constant rates

- $N_t=N_{t-1}-1+N_{birth}-N_{death}$
- N_{birth} ~ binomial (N, P_{birth})
- N_{death} ~ binomial (N, P_{death})
- Let's look at the code 'population_modeling_bdp.py'

Monte Carlo Simulation

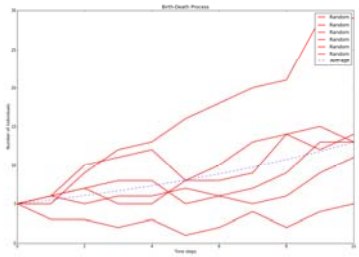
- Wrap the code by a loop
- Save results of each iteration



- p

Plot Results

- matplotlib, a library to illustrate your data



Leslie Model

$$\begin{bmatrix} n_0 \\ n_1 \\ \vdots \\ n_k \end{bmatrix}_{t+1} = \begin{bmatrix} F_o & F_1 & \cdots & F_k \\ S_0 & 0 & \cdots & 0 \\ 0 & \ddots & \cdots & 0 \\ 0 & 0 & S_{k-1} & 0 \end{bmatrix} \begin{bmatrix} n_0 \\ n_1 \\ \vdots \\ n_k \end{bmatrix}_t$$

```
for i in range(0, T):
    n=np.dot(l_m, n_o)
    n_o=n
    n_f[:,i]=n.squeeze()
```

Why squeeze?
Try n.ndim and n.squeeze().ndim

Let's look at the script 'population_modeling_lm.py'

10