OBJECT ORIENTED PROGRAMMING

Brief Recap

- Iterating over lists
- Methods for Manipulating Lists
- Methods for Manipulating Strings

Menu for Today!

- More on Lists
  - *Comparing List and String Objects*
  - *Aliasing*
- Introduction to Object Oriented Programming
  - *Data Abstraction*
  - *Python Classes*

SOME MORE ON LISTS

Comparing Lists and Strings

- In built notion of sequence
  - *Can access components but indexing*
  - *Slices work in the same way*
  - *+ and * operators work the same way*
  - *Iteration works in the same way*
- Lists are mutable, strings are immutable
  - `list[0] = 1` *is allowed*
  - `string[0] = '1'` *is not allowed*
  - *Be careful with aliasing when working with Lists*
- Elements
  - *Strings are homogenous: only contain characters*
  - *Lists can contain objects of any type, including other lists*

Converting Lists to Strings and Back

- `list(s)`: converts a string to a list
  - *every character from* `s` *is an element in the list*
- `s.split(char)`: split a string on a character parameter `char`, default is space
- `‘char'.join(L)`: converts a list of characters into a string
  - *The* `char` *parameter in quotes is added between every element*

```
list(‘Python’) -> ['P', 'y', 't', 'h', 'o', 'n’]
sentence = “One flew over the cuckoo’s nest”
sentence.split() -> ['One', 'flew', 'over', 'the', "cuckoo's", 'nest']
"".join(['a','b','c’]) -> ‘abc’
“_”.join (['a','b','c’]) -> 'a_b_c'
```

More on Aliasing

```
somePrimes = [3, 5, 7, 11, 13]
someOddNos = somePrimes
someOddNos.append(15)
```

- `someOddNos` is an alias for `somePrimes` – changing one changes the other!
- `append()` has a side effect

Cloning a List

```
somePrimes = [3, 5, 7, 11, 13]
someOddNos = somePrimes[:]
someOddNos.append(15)
```

- create a new list and copy every element using the slice `[:]`

Mutation and Iteration: Be Careful

- Avoid mutating a list as you are iterating over it

```
def remove_dups(L1, L2):
  for e in L1:
    if e in L2:
      L1.remove(e)
```

```
def remove_dups(L1, L2):
  L1_copy = L1[:]
  for e in L1_copy:
    if e in L2:
      L1.remove(e)
```

```
L1 = [1,2,3,4]
L2 = [1,2,5,6]
```

- L1 is `[2,3,4]` not `[3,4]` Why?
- Python uses an internal counter to keep track of index it is in the loop
- Mutating changes the list length but Python doesn't update the counter
- Loop never sees element 2

OBJECT ORIENTED PROGRAMMING

Class of Display Objects

- Imagine a class of objects: Displays
  - *All displays have two Boolean state variables*
    - On
    - Connected
  - *Each display device has it's own set of methods for switching on/off and connecting to a source*
- Projectors and TVs are both displays
  - *Their interfaces have somethings in common and some differences*
  - *Both have a method to "center" the projected picture on the display, but implementations can be very different*
- A TV remote will not work with a projector

Objects

- Python supports many different types of data

```
42    3.1416      ‘Toss a coin for your witcher’
[‘Witcher’, ‘Midnight Diner’, ‘7 Days in Entebbe’]
```

- So far, we have loosely said that every expression in Python evaluates to an “object”
- An object is said to be an instance of a type
  - `42` *is an instance of type* `int`
  - `‘Midnight Diner’` *is an instance of type* `string`

Object Oriented Programming

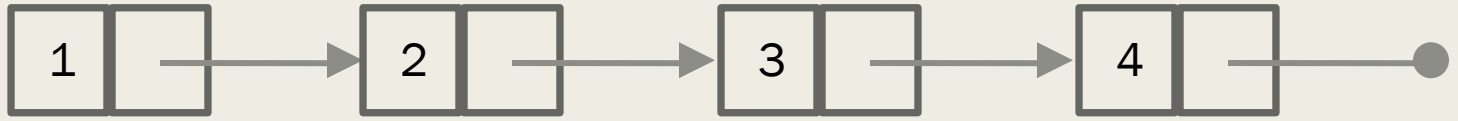
- EVERYTHING IN PYTHON IS AN OBJECT (and has a type)
  - *That includes objects of primitive Types such as* `int`, `float`, `bool`
  - *Also includes more complex Types such as* `string`, `list` *and* `function`
- What can we do with objects?
  - *We can create new objects of some type*
  - *We can manipulate objects*
  - *We can destroy objects*
- We have experience with creating and manipulating objects. What about destroying objects?
  - *Explicitly using del*
  - *Implicitly by loosing all references to an object. Python system will reclaim destroyed or inaccessible objects – called "garbage collection"*

Objects Allow Data Abstraction

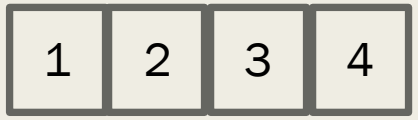
- Functions allowed us to abstract control
- Objects allow us to abstract details of data
- Formally, an object has the following attributes
  - *A type*
  - *An internal data representation – primitive or compound*
  - *A set of functions (methods) to interact with the object*

How Can We Represent Lists in Memory

- Linked List

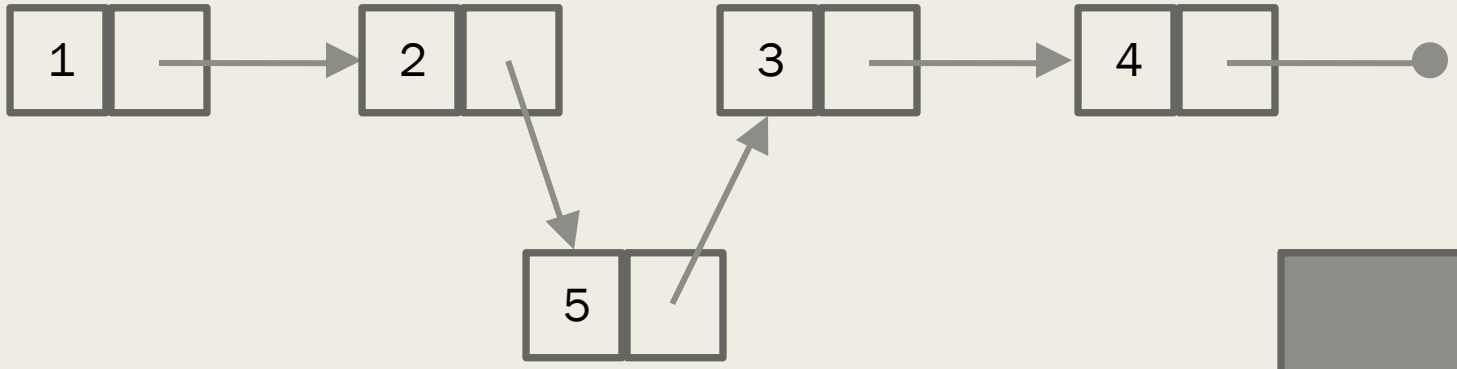


- Array

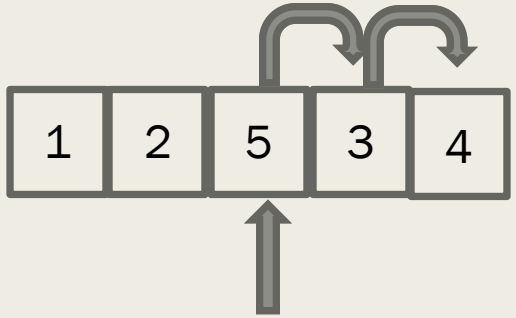


Inserting Elements:
`x.insert(2,5)`

- Linked List



- Array



Change in (internal) representation should not change the meaning of our programs

Data Abstraction

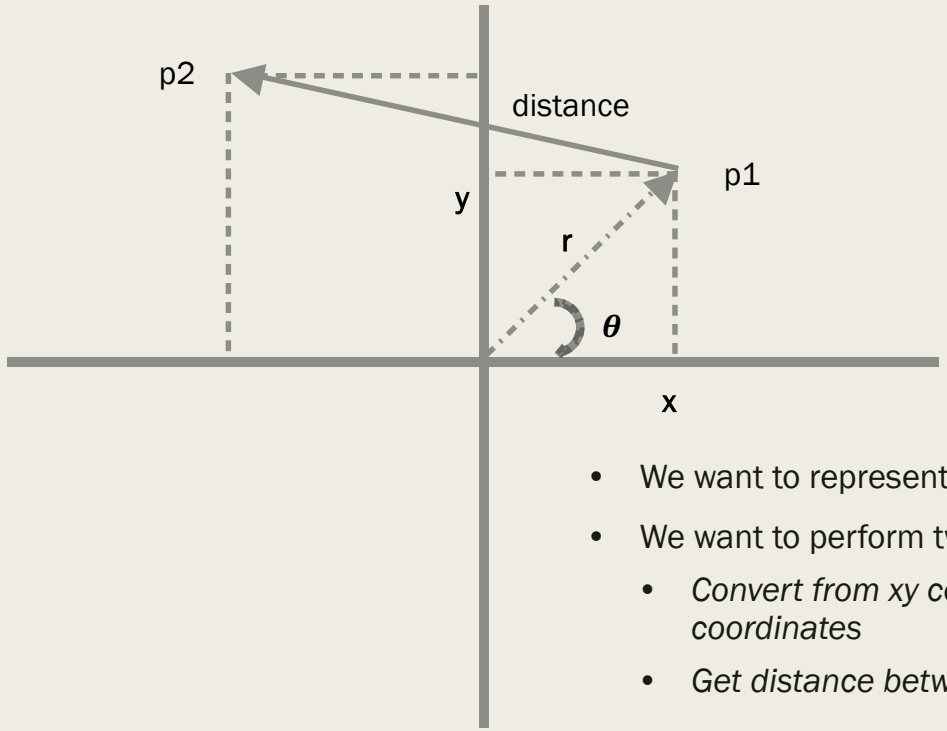
Data Abstraction

- Abstraction : Hide details
  - *Functions: hide details of control flow*
  - *Objects hide details of data data into packages together with procedures that work on them through well-defined interfaces*
- Divide-and-conquer development
  - *implement and test behavior of each class separately*
  - *increased modularity reduces complexity*
- Classes make it easy to reuse code
  - *many Python modules define new classes*
  - *each class has a separate environment (no collision on function names)*
  - *inheritance allows subclasses to redefine or extend a selected subset of a superclass' behavior*

Python Classes

- List objects
  - *Have a structure which can be manipulated with methods*
  - *Have type list*
- Can we create our own custom objects?
  – How do we define the internal structure of a custom object – How do we define the methods to manipulate the object
  – What type should it have?
- We need to define a Python class
  – A Python class is like a custom `type` *that we have defined*

2 Dimensional Coordinate System



- We want to represent points on a plane as a class
- We want to perform two operations:
  - *Convert from xy coordinates to polar coordinates*
  - *Get distance between two points*

Class Of 2D Points

```
import math
class point:
    def __init__(self,x,y):
        self.x = x
        self.y = y
    def distance(self, other):
        return ((self.x-other.x)**2 + (self.y-other.y)**2)**0.5
```

Class of 2D Points

```
def getPolar(self):
    if (self.x > 0):
        theta = math.atan(self.y / self.x)
    elif (self.x < 0):
        theta = math.atan(self.y / self.x) + math.pi
    else:
        theta = math.pi
    r = (self.x**2 + self.y**2)**0.5
    return(r, theta)
```