Events & Objects

… delegates, encapsulation, polymorphism, inheritance

Objects

In the 1st lab we identified some patterns.

Things the computer is good at.

The original idea of patterns from Chris Alexander was a language for people to discuss independent of how experienced they are.

Sequence, Repetition, Selection, Exceptions … are all patterns.

Today we will add delegates, encapsulation, polymorphism and inheritance.

Design Patterns -> Extreme Programming

Kent Beck, Ward Cunningham and Ron Jeffries formulated extreme Programming in 1999.

Beck wrote his book, Implementation patterns after XP and the agile manifesto.

Even though Agile and XP don’t advocate big design up front the design patterns still seem relevant.

The patterns that we are discussing are foundational.

Delegate

A Handler for an Event

Sometimes called publish and subscribe

```
# we make a flask object
app = Flask(__name__)

# the sms service calls this "web hook"
@app.route("/sms",  methods=['POST'])
def sms():
        nPhone = request.form["from"]
        sMessage = request.form["body"]
        return Response("<Response><Message>"
+ sMessage + " sms from " + nPhone + " answered
here</Message></Response>",
mimetype='text/xml')

# a route where we will display a welcome
message via an HTML template
@app.route("/")
def server():
        return render_template('index.html')
```

Encapsulation

```
class Game:
        def __init__(self):
        self.__nCurrent = -1
        def takeTurn(self, sInput):
        if(self.__nCurrent == -1):
        self.__nCurrent = 0
        return["Welcome to I know
you are",
                "This is an annoying
game you might have played",
                "with your sibling."
                ]
        else:
        return["I know you are a",
sInput, "But what am I?"]
```

Mix behaviour and data

Polymorphism

The same stimulus or method elicits different behaviour.

```
        def takeTurn(self, sInput):
        if(self.__nCurrent == -1):
        self.__nComputer = random.randint(1,100)
        self.__nCurrent = 0
        return["Welcome to Over and Unders",
                "The computer is thinking of a
number between 1 and 99.",
                "Try to guess in as few tries as
possible."
                ]
        else:
        try:
                nInput = int(sInput)
                if nInput > self.__nComputer:
                return[
                        "Too High"
                ]
```

Inheritance

```
class Game:
        def __init__(self):
        self.__isDone = False
        def isDone(self,
isDone=False):
        if isDone:
        self.__isDone = isDone
        return self.__isDone
```

Also generalization and specialization

These 4 Ideas

Take us to modern programming

And Away from the command line

We will look for more patterns as we dig into Django and test driven development