Week 9

CSC111 — Spring 2015

Dominique Thiébaut
dthiebaut@smith.edu
Quick Words about Matthew Jockers' "Man in the Hole" story patterns, and Digital Humanities
Dealing with Exceptions (Chapter 7.4)

Defining Classes (Chapter 10)
```python
# getInput: returns an integer larger
# than 0. Expected to be robust…
def getInput():
    x = int( input( "Enter a positive int: " ) )
    while x < 0:
        x = int( input( "Invalid number: Please try again: " ) )
    return x

def main():
    num = getInput()
    print( "you entered", num )

main()
```
```python
# getIn: returns an integer larger
# than 0. Expected to be robust...
def getIn():
    x = int( input( "Enter a positive int: " ) )
    while x < 0:
        x = int( input( "Invalid number: Please try again: " ) )
    return x

def main():
    num = getIn()
    print( "You entered", num )

main()
```

```
Python 3.1.1 (r311:74543, Aug 24 2009, 18:44:04)
[GCC 4.0.1 (Apple Inc. build 5493)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> 
```

```
Enter a positive int: -
Traceback (most recent call last):
  File "/Users/thiebaut/Desktop/Dropbox/111/Week9/exceptions1.py", line 13, in <module>
    main()
  File "/Users/thiebaut/Desktop/Dropbox/111/Week9/exceptions1.py", line 10, in main
    num = getIn()
  File "/Users/thiebaut/Desktop/Dropbox/111/Week9/exceptions1.py", line 4, in getIn
    x = int( input( "Enter a positive int: " ) )
ValueError: invalid literal for int() with base 10: '-'
>>> 
```
```python
# get_input: returns an integer larger than 0. Expected to be robust...
def get_input():
    x = int(input("Enter a positive int: ") )
    while x < 0:
        x = int(input("Invalid number: Please try again: ") )
    return x

def main():
    num = get_input()
    print("You entered", num)

main()
```

```
>>> Enter a positive int: -
Traceback (most recent call last):
  File "/Users/thiebaut/Desktop/Dropbox/111/Week9/exceptions1.py", line 13, in main
    main()
  File "/Users/thiebaut/Desktop/Dropbox/111/Week9/exceptions1.py", line 10, in main
    num = get_input()
  File "/Users/thiebaut/Desktop/Dropbox/111/Week9/exceptions1.py", line 4, in get_input
    x = int(input("Enter a positive int: ") )
ValueError: invalid literal for int() with base 10: '-'
```
The OS manages the resources of a computer

Whenever an operation is illegal, the OS generates an Exception to flag the fault
Some exceptions are generated by the Python interpreter.
While Exceptions are a pain in the neck, they are an integral part of programming.
Try/Except Statement

```python
try:
    python code that might generate an exception

Except exceptionXYZ:
    python code to run in case there's an exception
```
# getInputs2: returns an integer larger
# than 0. Prompts the user to keep inputting as long
# as the input is not a positive number. Will not crash
# on string or floats
def getInputs2():
    while True:
        try:
            x = int( input( "Enter a positive int: " ) )
            if x > 0:
                return x
            else:
                print( "Invalid number. Try again." )
        except ValueError:
            print("Not an integer, try again."")

def main():
    num = getInputs2()
    print("You entered", num )

main()
Approach to Handling Exceptions

1. Run code **without** try/except statements

2. Test thoroughly

3. Fix whatever can be fixed with "regular" python code

4. Record all exceptions that cannot be fixed otherwise, and add **try/except** to catch them
Multiple Exceptions
(taken from Zelle)
Solutions to: $a*x^2 + b*x + c = 0$

```
import math

def ZelleExample():
    print( "solution for quadratic equation" )
    try:
        a, b, c = eval( input( "enter 3 coefficients ( a, b, c ) " ) )
        disc = math.sqrt( b*b - 4*a*c )
        root1 = (-b + disc) / (2*a)
        root2 = (-b - disc) / (2*a)
        print( "solutions: ", root1, root2 )
    except NameError:
        print( "You didn't enter 3 numbers" )
    except TypeError:
        print( "your input were not all numbers" )
    except SyntaxError:
        print( "Forgot a comma between the numbers?" )
    except ValueError:
        print( "No real roots, negative discriminant" )
    except:
        print( "Something went wrong..." )
```
Hardening the Function

```python
def ZelleExample():
    import math
    print( "solution for quadratic equation" )
    try:
        a, b, c = eval( input( "enter 3 coefficients ( a, b, c ) " ) )
        disc = math.sqrt( b*b - 4*a*c )
        root1 = (-b + disc) / (2*a)
        root2 = (-b - disc) / (2*a)
        print( "solutions: ", root1, root2 )
        return True
    except NameError:
        print( "You didn't enter 3 numbers" )
    except TypeError:
        print("your input were not all numbers")
    except SyntaxError:
        print( "Forgot a comma between the numbers?" )
    except ValueError:
        print( "No real roots, negative discriminant" )
    except:
        print("Something went wrong... ")
        return False
```
Dealing with Exceptions (Chapter 7.4)

Defining Classes (Chapter 10)
Dealing with Exceptions (Chapter 7.4)

Defining Classes (Chapter 10)
Coding Dice
Using the Objects

# Create 2 dice, one with 6 sides, one with 8
d1 = Die( 6 )
d2 = Die( 8 )

# Roll both dice
d1.roll()
d2.roll()

# display their value
print( "Die 1: ", d1.getValue() )
print( "Die 2: ", d2.getValue() )
We need to create the blueprint for a box…

(object)

roll()
We need to create the blueprint for the box…

getValue()
object

blueprint
object

object

blueprint

Blueprint
class def
blueprint = class
# libraries
import random

# a class for a die
class Die:
    def __init__(self, n):
        self.noSides = n
        self.value = 1

    def roll(self):
        self.value = random.randrange(1, self.noSides+1)

    def getValue(self):
        return self.value
# libraries
import random

# a class for a die
class Die:
    def __init__(self, n):
        self.noSides = n
        self.value = 1

    def roll(self):
        self.value = random.randrange(1, self.noSides+1)

    def getValue(self):
        return self.value

constructor
# libraries
import random

# a class for a die
class Die:
    def __init__(self, n):
        self.noSides = n
        self.value = 1

    def roll(self):
        self.value = random.randrange(1, self.noSides+1)

    def getValue(self):
        return self.value

# Create 2 dice, one with 6 sides
d1 = Die(6)
d2 = Die(8)

# Roll both dice
d1.roll()
d2.roll()

# display their value
print( "Die 1: ", d1.getValue() )
print( "Die 2: ", d2.getValue() )
A Die Class

# libraries
import random

# a class for a die
class Die:
    def __init__( self, n ):
        self.noSides = n
        self.value = 1

    def roll( self ):
        self.value = random.randrange( 1, self.noSides+1 )

    def getValue( self ):
        return self.value

# Create 2 dice, one with 6 sides
d1 = Die( 6 )
d2 = Die( 8 )

# Roll both dice
d1.roll()
d2.roll()

# display their value
print( "Die 1: ", d1.getValue() )
print( "Die 2: ", d2.getValue() )
# A Die Class

```python
# libraries
import random

# a class for a die
class Die:
    def __init__(self, n):
        self.noSides = n
        self.value = 1

    def roll(self):
        self.value = random.randrange(1, self.noSides+1)

    def getValue(self):
        return self.value
```

reference to the object
A Die Class

# Create 2 dice, one with 6 sides
d1 = Die( 6 )
d2 = Die( 8 )

# Roll both dice
d1.roll()
d2.roll()

# display their value
print( "Die 1: ", d1.getValue() )
print( "Die 2: ", d2.getValue() )

# libraries
import random

# a class for a die
class Die:
    def __init__( self, n ):
        self.noSides = n
        self.value = 1

    def roll( self ):
        self.value = random.randrange( 1, self.noSides+1 )

    def getValue( self ):
        return self.value
# libraries
import random

# a class for a die
class Die:
    def __init__(self, n):
        self.noSides = n
        self.value = 1
    def roll(self):
        self.value = random.randrange(1, self.noSides+1)
    def getValue(self):
        return self.value

# Create 2 dice, one with 6 sides
d1 = Die(6)
d2 = Die(8)

# Roll both dice
roll(d1)
roll(d2)

# display their value
print("Die 1: ", d1.getValue())
print("Die 2: ", d2.getValue())
# libraries
import random

# a class for a die
class Die:
    def __init__(self, n):
        self.noSides = n
        self.value = 1

    def roll(self):
        self.value = random.randrange(1, self.noSides+1)

    def getValue(self):
        return self.value

makes the variable a "member" of the object
Exercise

Write a program that maintains a list of objects that are *cats*. Cats have a *name*, have a *breed*, may or may not be *vaccinated*, and have an *age* expressed in years.
Examples

Minou, 3, vac, stray
Max, 1, not-vac, Burmese
Gizmo, 2, vac, Bengal
Garfield, 4, not-vac, Orange Tabby
Using Cat Objects

```python
# Minou, 3, vac, tat, stray
cat1 = Cat("Minou", True, "stray", 3)

if cat1.isVaccinated():
    print(cat1.getName(), "is vaccinated")
else:
    print(cat1.getName(), "is not vaccinated")
```
Wanted:

A program that
- outputs all the cats
- outputs only the vaccinated cats
- outputs the cats 2 or older
We stopped here last time...
Pair Programming in Lab 9

Review of Classes and Objects

Cats, Cats, Cats...
  Default string representation
List of Cats
Reading CSV Files of Cats
Searching for a Cat in a List
Pair Programming

Laurie A. Williams and Robert R. Kessler

All I Really Need To Know About Pair Programming, I Learned in Kindergarten

When it comes to programming practices, studies show two heads are almost always better than one.

Pair programming is a practice in which two programmers work side-by-side at one computer, continuously collaborating on the same design, algorithm, code, or test. This method has been demonstrated to improve productivity and the quality of software products. Moreover, a recent survey (hereafter referred to as the “pair programming survey”) found that programmers were universally more confident in their solutions when programming in pairs as opposed to working alone. Likewise, 96% agreed that the IoT programming was more fun when pair programming.

However, many programmers are long conditioned to working alone and often resist the transition to pair programming. Ultimately, most make this transition with great success. The goal of this article is to help programmers become effective pair programmers. The transition to pair programming often involves practicing everyday civility, as illustrated in an essay by Robert Fulghum (see box). Here, we take each line from the essay (with occasional poetic license) to explore the inherent lessons related to successful pair programming.

Anecdotal and initial statistical evidence indicates that pair programming is highly beneficial. In extreme programming (XP)—an emerging software development methodology—all production code is written with a partner. XP was developed initially by Smalltalk code developer and consultant Kent Beck with colleagues Ward Cunningham and Ron Jeffries. XP’s requirements gathering, resource allocation, and design practices are a radical departure from most accepted methodologies. Customer requirements are written as fairly informal “User Story” cards where a
Pair Programming

- Article by Laurie Williams and Robert Kessler, 2000
- Appears in *Communications of the ACM*
- Recommends that programmers work in pairs, at 1 computer.

**Benefits:**
- fewer bugs
- faster development time--twice as fast,
- twice as many solutions explored
Pair Programming

• Recommendations:
  • 70% time = development
  • 30% time = breaks + other activities
Pair Programming in Lab 9

Review of Classes and Objects

Cats, Cats, Cats...
Default string representation
List of Cats
Reading CSV Files of Cats
Searching for a Cat in a List
Review
Review

Blueprint
class def
Review

Blueprint
class def
Review
Review

Blueprint
class def

value
Instantiation: Object is instance of a Class

member variable

method

Class
# libraries
import random

# a class for a die
class Die:
    def __init__(self, n):
        self.noSides = n
        self.value = 1

    def roll(self):
        self.value = random.randrange(1, self.noSides+1)

    def getValue(self):
        return self.value

# Create 2 dice, one with 6 sides
d1 = Die(6)
d2 = Die(8)

# Roll both dice
d1.roll()
d2.roll()

# display their value
print("Die 1: ", d1.getValue())
print("Die 2: ", d2.getValue())
Pair Programming in Lab 9

Review of Classes and Objects

Cats, Cats, Cats...

- Default string representation
- List of Cats
- Reading CSV Files of Cats
- Searching for a Cat in a List
Back to Cats
Using Cat Objects

```python
# Minou, 3, vac, tat, stray
cat1 = Cat( "Minou", True, "stray", 3 )

if cat1.isVaccinated():
    print( cat1.getName(), "is vaccinated" )
else:
    print( cat1.getName(), "is not vaccinated" )
```
Step 1: Implement the Class

class Cat:
def __init__(self, name, vacc, breed, age):
    self.name = name
    self.vacc = vacc
    self.breed = breed
    self.age = age
return

def getName(self):
    return self.name

def isVaccinated(self):
    return self.vacc

def __str__(self):
    if self.vacc:
        return "Vaccinated"
    else:
        return "Not Vaccinated"

def main():
    # Minou, 3, vaccinated
    cat1 = Cat("Minou", True, "Burmese", 3)
    if cat1.isVaccinated():
        print("Vaccinated")
    else:
        print("Not Vaccinated")
    # Silky, 2, vaccinated
    cat2 = Cat("Silky", True, "Burmese", 2)
    if cat2.isVaccinated():
        print("Vaccinated")
    else:
        print("Not Vaccinated")
Step 2: Create a List of Cats
Step 3: Read a CSV File of Cats
Step 4: Display Only Vaccinated Cats
We stopped here last time...
Looping through a list of objects

Doing graphics with objects
Step 5: Search for the Youngest Cat
Looping through a list of objects

Doing graphics with objects
Graphic Cars Moving Around
Car Geometry

[Diagram of a car with dimensions labeled: 140, 40, 180, 60, 20.]
Video 1

https://www.youtube.com/watch?v=4GxPrESfdnM
Video 2

https://www.youtube.com/watch?v=3wjCwtc_hk