Week 9
CSC111 - Fall 2018
• Dealing with Exceptions (Chapter 7.4)

• Defining Classes (Chapter 10)
# getInput: returns an integer larger
# than 0. Expected to be robust...

def getInput():
    while True:
        x = eval(input("Enter a positive int: "))
        if x >= 0:
            return x
        print("Invalid number: Please try again: ")

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

main()
```python
# getInput: returns an integer larger
# than 0.

def getInput():
    while True:
        x = eval( input( "Enter a positive int: " ) )
        if x >= 0:
            return x
    print( "You entered a negative integer." )

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

main()
```

```
>>> Enter a positive int: 
SyntaxError: unexpected EOF while parsing
```
# getInput: returns an integer larger
# than 0.
def getInput():
    while True:
        x = eval(input("Enter a positive int: ") )
        if x >= 0:
            return x
        print("You entered a negative integer."

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

main()
Some exceptions are generated by the Python interpreter.
RAM

Operating System

Python Program

Exception

Processor

Keyboard

Disk

Network
If Exception generated by Python Interpreter, OS "kills" the Python Program
We want to prevent exceptions from going all the way up to the operating system…
• We want to "isolate" dangerous code areas
• We want a tight isolation blocks around the potentially dangerous code sections
Coding exceptions is a pain in the neck but exceptions are an integral part of programming
Try/Except Statement

try:

```python
don't write me
```

Python code that might generate an exception

Except `exceptionXYZ`:

```python
don't write me
```

Python code to run in case there's an exception
# get input: returns an integer larger
# than 0.
def getInput():
    while True:
        try:
            x = eval(input("Enter a positive int: "))
        except SyntaxError:
            print("Invalid input. Try again!")
            continue
        if x >= 0:
            return x
        print("You entered a negative integer.")

def main():
    num = getInput()
    print("num = ", num)
main()
# get input: returns an integer larger than 0.
def getInput():
    while True:
        try:
            x = eval(input("Enter a positive int: "))
        except SyntaxError:
            print("Invalid input. Try again!"
        continue
    if x >= 0:
        return x
    print("You entered a negative integer.")

def main():
    num = getInput()
    print("num = ", num)
main()
def getInput():
    while True:
        try:
            x = eval(input("Enter a positive int: "))
        except SyntaxError:
            print("Invalid input. Try again!")
            continue
        except NameError:
            print("Invalid input. Not an integer")
            continue
        if x >= 0:
            return x

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.

5. Make the **try** section as **small** as possible.
Multiple Exceptions (taken from Zelle)
Solving 2nd Degree Equations $ax^2 + bx + c = 0$

Use the quadratic formula (QF)

The roots for the equation $ax^2 + bx + c = 0$ are

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$b^2 - 4ac$ is called the *discriminant* because its value indicates what type of roots there are. Specifically, if $b^2 - 4ac$ is a perfect square, we have fractional roots, if $b^2 - 4ac < 0$ there is no real roots.
Solving Equation of Degree 2

```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)
    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

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**2 - 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)

CSV Files

Defining Classes (Chapter 10)
• **Comma-Separated Values**

• Very popular way of representing information where all records have the same format

• Easy to implement

• Examples: [https://people.sc.fsu.edu/~jburkardt/data/csv/csv.html](https://people.sc.fsu.edu/~jburkardt/data/csv/csv.html)
<table>
<thead>
<tr>
<th>Name</th>
<th>Sex</th>
<th>Age</th>
<th>Height (in)</th>
<th>Weight (lbs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alex</td>
<td>M</td>
<td>41</td>
<td>74</td>
<td>170</td>
</tr>
<tr>
<td>Bert</td>
<td>M</td>
<td>42</td>
<td>68</td>
<td>166</td>
</tr>
<tr>
<td>Carl</td>
<td>M</td>
<td>32</td>
<td>70</td>
<td>155</td>
</tr>
<tr>
<td>Dave</td>
<td>M</td>
<td>39</td>
<td>72</td>
<td>167</td>
</tr>
<tr>
<td>Elly</td>
<td>F</td>
<td>30</td>
<td>66</td>
<td>124</td>
</tr>
<tr>
<td>Fran</td>
<td>F</td>
<td>33</td>
<td>66</td>
<td>115</td>
</tr>
<tr>
<td>Gwen</td>
<td>F</td>
<td>26</td>
<td>64</td>
<td>121</td>
</tr>
<tr>
<td>Hank</td>
<td>M</td>
<td>30</td>
<td>71</td>
<td>158</td>
</tr>
<tr>
<td>Ivan</td>
<td>M</td>
<td>53</td>
<td>72</td>
<td>175</td>
</tr>
<tr>
<td>Jake</td>
<td>M</td>
<td>32</td>
<td>69</td>
<td>143</td>
</tr>
<tr>
<td>Kate</td>
<td>F</td>
<td>47</td>
<td>69</td>
<td>139</td>
</tr>
<tr>
<td>Luke</td>
<td>M</td>
<td>34</td>
<td>72</td>
<td>163</td>
</tr>
<tr>
<td>Myra</td>
<td>F</td>
<td>23</td>
<td>62</td>
<td>98</td>
</tr>
<tr>
<td>Neil</td>
<td>M</td>
<td>36</td>
<td>75</td>
<td>160</td>
</tr>
<tr>
<td>Omar</td>
<td>M</td>
<td>38</td>
<td>70</td>
<td>145</td>
</tr>
<tr>
<td>Page</td>
<td>F</td>
<td>31</td>
<td>67</td>
<td>135</td>
</tr>
<tr>
<td>Quin</td>
<td>M</td>
<td>29</td>
<td>71</td>
<td>176</td>
</tr>
<tr>
<td>Ruth</td>
<td>F</td>
<td>28</td>
<td>65</td>
<td>131</td>
</tr>
</tbody>
</table>
# readCSVFile.py
# D. Thiebaut
# Example of a program that reads a CSV file
# and displays some of its contents.

def readCSV( fileName ):
    file = open( fileName, 'r' )
    Records = [ ]
    lines = file.readlines()
    for line in lines:
        words = line.strip().split( ',', '' )
        Records.append( words )
    file.close()
    return Records

def main():
    fileName = input( "File name? " )
    recs = readCSV( fileName )
    for i in range( len( recs ) ):
        record = recs[i]
        # record is a tuple
        # join each word in the tuple by a tab, and print
        # resulting string
        print( "Record ", i, "=", "\t".join( record ) )

main()
CSV and MS Excel

What type of file do you want to import?

- CSV file
  Select this file type when you want to import text files that contain comma-separated values. Most financial institutions offer this format for saving account activity.

- FileMaker Pro database
  Select this file type when you want to import data from an .fp5 or .fp7 database created with FileMaker Pro. You can import all records in the database or specify.

- HTML file
  Select this file type when you want to import information from an HTML file.

- Text file
  Select this file type when you want to import text files. This option works if your files contain values separated by tabs or spaces.

More about how to import data...

Text Import Wizard - Step 1 of 3

The Text Wizard has determined that your data is Fixed Width.

If this is correct, choose Next, or choose the Data Type that best describes your data.

Original data type

Choose the file type that best describes your data:

- Delimited
  - Characters such as commas or tabs separate each field.

- Fixed width
  - Fields are aligned in columns with spaces between each field.

Start import at row: 1 File origin: Macintosh

Data preview


<table>
<thead>
<tr>
<th>Name</th>
<th>Sex</th>
<th>Age</th>
<th>Height (in)</th>
<th>Weight (lbs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alex</td>
<td>M</td>
<td>41</td>
<td>74</td>
<td>170</td>
</tr>
<tr>
<td>Bert</td>
<td>M</td>
<td>42</td>
<td>68</td>
<td>166</td>
</tr>
<tr>
<td>Carl</td>
<td>M</td>
<td>32</td>
<td>70</td>
<td>155</td>
</tr>
<tr>
<td>Dov</td>
<td>M</td>
<td>39</td>
<td>72</td>
<td>167</td>
</tr>
<tr>
<td>Elly</td>
<td>F</td>
<td>38</td>
<td>66</td>
<td>124</td>
</tr>
</tbody>
</table>
CSV and MS Excel

<table>
<thead>
<tr>
<th>Name</th>
<th>Sex</th>
<th>Age</th>
<th>Height (in)</th>
<th>Weight (lbs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alex</td>
<td>M</td>
<td>41</td>
<td>74</td>
<td>170</td>
</tr>
<tr>
<td>Bert</td>
<td>M</td>
<td>42</td>
<td>68</td>
<td>166</td>
</tr>
<tr>
<td>Carl</td>
<td>M</td>
<td>32</td>
<td>70</td>
<td>155</td>
</tr>
<tr>
<td>Dave</td>
<td>M</td>
<td>39</td>
<td>72</td>
<td>167</td>
</tr>
<tr>
<td>Elly</td>
<td>F</td>
<td>30</td>
<td>66</td>
<td>124</td>
</tr>
<tr>
<td>Fran</td>
<td>F</td>
<td>33</td>
<td>66</td>
<td>115</td>
</tr>
<tr>
<td>Gwen</td>
<td>F</td>
<td>26</td>
<td>64</td>
<td>121</td>
</tr>
<tr>
<td>Hank</td>
<td>M</td>
<td>30</td>
<td>71</td>
<td>158</td>
</tr>
<tr>
<td>Ivan</td>
<td>M</td>
<td>53</td>
<td>72</td>
<td>175</td>
</tr>
<tr>
<td>Jake</td>
<td>M</td>
<td>32</td>
<td>69</td>
<td>143</td>
</tr>
<tr>
<td>Kate</td>
<td>F</td>
<td>47</td>
<td>69</td>
<td>139</td>
</tr>
<tr>
<td>Luke</td>
<td>M</td>
<td>34</td>
<td>72</td>
<td>163</td>
</tr>
<tr>
<td>Myra</td>
<td>F</td>
<td>23</td>
<td>62</td>
<td>98</td>
</tr>
<tr>
<td>Neil</td>
<td>M</td>
<td>36</td>
<td>75</td>
<td>160</td>
</tr>
<tr>
<td>Omar</td>
<td>M</td>
<td>38</td>
<td>70</td>
<td>145</td>
</tr>
<tr>
<td>Page</td>
<td>F</td>
<td>31</td>
<td>67</td>
<td>135</td>
</tr>
<tr>
<td>Quin</td>
<td>M</td>
<td>29</td>
<td>71</td>
<td>176</td>
</tr>
<tr>
<td>Ruth</td>
<td>F</td>
<td>28</td>
<td>65</td>
<td>131</td>
</tr>
</tbody>
</table>
Dealing with Exceptions (Chapter 7.4)

CSV Files

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

CSV Files

Defining Classes (Chapter 10)
Coding Dice
Using the Objects

```python
# 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 → blueprint → object

Blueprint
class def
object

object

object

blueprint = class
# libraries
from random import randrange

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

    def roll(self):
        self.value = 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

A Die Class
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())
# 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())
# 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
# 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())
# 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
# 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
Playing dice...
We stopped here last time...
# libraries
import random

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

def getValue(self):
    return self.value

class Die:
    def __init__(self, n):
        self.noSides = n
        self.value = 1

# 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())

# a class for a die

from dieClass import Die

playDice.py

dieClass.py
Why Create a Die Class? Randint Could have Sufficed

• **Modularity**

• Details are hidden *(Information hiding)*

• The Die class can easily be *enhanced/modified* without having to change main program
  
  • die with a bias
  
  • history of rolls
  
  • keeping track of stats
Write a program that maintains a list of cat objects. Cats have a name, a breed, may or may not be vaccinated and have an age expressed in years (float).
Examples

Minou, 3, vac, stray
Max, 1, not-vac, Burmese
Gizmo, 2, vac, Bengal
Garfield, 4, not-vac, Orange Tabby
# Minou, 3, vac, 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
• 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
Important Concepts:

LOCAL vs. GLOBAL
What can you say about this program? Focus on the variable a...
What can you say about this program?
Focus on the variable `a`...
What can you say about this program?
Focus on the variable `a`...
What can you say about this program? Focus on the variable `a`...
What can you say about this program? Focus on the variable a...
```python
a = 3

def func1( x ):
    print( x * a )

def func2( y ):
    print( y * a )

def main():
    func1( 10 )
    func2( 10 )

main()```

```
30
30
```
Global Variable

def func1(x):
    print(x * a)

def func2(y):
    print(y * a)

def main():
    func1(10)
    func2(10)

main()
What can you say about this program? Focus on the variable `a`...
a = 3

def func1( x ):  
    print( x * a )

def func2( y ):  
    a = 8  
    print( y * a )

def main():  
    func1( 10 )  
    func2( 10 )  
    print( a )

main()

30
80
3
```python
a = 3

def func1(x):
    print(x * a)

def func2(y):
    a = 8
    print(y * a)

def main():
    func1(10)
    func2(10)
    print(a)

main()
```

Global Variable: `a = 3`

Local Variable: `a = 8`
What can you say about this program?
Focus on the variable a...
```python
a = 3

def func1(x):
    print(x * a)

def func2(y):
    global a
    a = 8
    print(y * a)

def main():
    func1(10)
    func2(10)
    print(a)
```

```
30
80
8
```
Blueprint
class def
Blueprint
class def
Review
Review
Instantiation: Object is instance of a Class

member variable

method
# libraries
import random

d1 = Die( 6 )
d2 = Die( 8 )

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

# 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
# Minou, 3, vaccinated, 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
    
def __str__(self):
        s = (self.name + ' (' + self.breed + ') is ')
        if self.vacc:
            s += 'vaccinated
        else:
            s += 'not vaccinated
        return s

    def isVaccinated(self):
        return self.vacc

    def getName(self):
        return self.name

    def main():
        # Minou, 3, vaccinated
        cat1 = Cat('Minou', True, 'Persian', 3)
        if cat1.isVaccinated():
            print('Minou is vaccinated')
            print('Minou (Persian), not vaccinated, 3 yrs old.
        else:
            print('Minou (stray), not vaccinated, 3 yrs old.

        # Silky, 2, not vaccinated
        cat2 = Cat('Silky', False, 'Sphynx', 2)
        if cat2.isVaccinated():
            print('Silky is vaccinated')
            print('Silky (Sphynx), vaccinated, 2 yrs old.
        else:
            print('Silky (Sphynx), not vaccinated, 2 yrs old.'
Step 2: Create a List of Cats

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

def getName(self):
    return self.name

def isVaccinated(self):
    return self.vaccinated

def __str__(self):
    s = self.name + ' (' + self.breed + '), ' + self.age + ' yrs old.
    return s

def main():
# Minou, 3, not vaccinated
# Silky, 2, vaccinated
Minou (stray), not vaccinated, 3 yrs old.
Silky (Siamese), vaccinated, 2 yrs old.
>>> 

Python 3.7.1 (default, Dec 23 2018, 21:15:48) 
[GCC 7.3.0] on darwin
Type "copyright", "credits" or "license()" for more information.

>>> 

>>> 

== RESTART: /Users/thiebaut/Desktop/Dropbox/111/Week0/cats0.py ==

>>> 

>>> 

>>> 

>>> 

>>> 

>>> 

>>> 

>>> 

>>> 

>>> 

>>> 

>>> 

>>> 

>>> 

>>> 

>>> 

>>> 

>>> 

>>> 

>>> 

>>> 

>>> 

>>> 

>>> 

>>> 

>>> 

>>> 

>>> 

>>> 

>>> 

>>> 

>>> 

>>> 

>>>
Step 3: Read a CSV File of Cats
Step 4: Display Only Vaccinated Cats

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

  def is_vaccinated(self):
    return self.is_vaccinated

  def __str__(self):
    s = "{} ({}), {} yo, breed: {}
".format(self.name, "" if self.is_vaccinated else "not ", self.age, self.breed)
    return s

def main():
  # Minou, 3, stray
  cat1 = Cat("Minou", False, 3, "stray")
  if cat1.is_vaccinated:
    print("Minou is vaccinated")
  else:
    print("Minou is not vaccinated")

  # Silky, 2, Burmese
  cat2 = Cat("Silky", True, 2, "Burmese")
  if cat2.is_vaccinated:
    print("Silky is vaccinated")
  else:
    print("Silky is not vaccinated")

main()
Step 5: Search for the Youngest Cat

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

def getName(self):
    return self.name

def isVaccinated(self):
    return self.vaccinated

def __str__(self):
    s = ""
    s += self.breed + " " + str(self.age) + " yrs old.
    if self.vaccinated:
        s += "Vaccinated"
    else:
        s += "Not Vaccinated"
    return s

def main():
    # Minou, 3, Burmese, vaccinated
    cat1 = Cat("Minou", 3, "Burmese", True)
    if cat1.isVaccinated():
        print("Minou is vaccinated")
    else:
        print("Minou is not vaccinated")

    # Silky, 2, Burmese, not vaccinated
    cat2 = Cat("Silky", 2, "Burmese", False)
    if cat2.isVaccinated():
        print("Silky is vaccinated")
    else:
        print("Silky is not vaccinated")
• Looping through a list of objects

• Object-Oriented Graphics
Graphic Cars Moving Around
Car Geometry

![Car Geometry Diagram](image)
Inspiration...
https://www.youtube.com/watch?v=4GxPrESfdnM
Crosswalk - Low density
16.5 ms per timestep

https://www.youtube.com/watch?v=pqBSNAOsMDc
https://www.youtube.com/watch?v=3wjCwtc_-hk