PYTHON CRASH COURSE

http://eddiema.ca/py

LECTURE 3

```
def tell_me_if_i_am_crazy(i_see_a_python):
    if i_see_a_python:
        print 'Quite crazy.'
    else:
        print 'Less crazy.'
```

EDDIE MA

JANUARY 2010

http://eddiema.ca

Oh right-- Python Documentation

http://docs.python.org/

Python's official documentation.

It's very comprehensive!

Take care of the version of Python.

Agenda

A few more basics, then input/output

- Finishing Loops
  - `break`
  - `continue`
- Two Final Datatypes
  - Dictionaries and Sets
- Exceptions
  - "Illogical" Flow
  - `try`, `except`, `finally`
  - ~~`raise`~~
- File handling
  - `open()`, `close()`, `read()`, `write()`
- Cheating with Pipes
  - `sys.stdin`
- Console input
  - `raw_input()`, `input()`
- Command Line Args
  - `sys.argv`
- Regular Expressions
  - `re`

`break` and `continue`

Final keywords about loops

- Use the `break` keyword to cause your program to immediately leave a loop
- In the example to the right, a `while` loop collects 500 random dice rolls
- The loop is broken before 500 rolls if the die rolls a 3.
- The list is printed afterward

```
from random import randint
alist = []
while len(alist) < 500:
    diceroll = randint(1, 6)
    if diceroll == 3:
        break
    alist.append(diceroll)
print alist
```

~~`break`~~ and `continue`

The zen of `continue`: pass go without collecting $200

- `continue` is used to cause the loop to immediately repeat without finishing its current repetition
- In this example, 500 dice rolls are always appended to the list, but dice rolls of 3 are discarded.

```
from random import randint
alist = []
while len(alist) < 500:
    diceroll = randint(1, 6)
    if diceroll == 3:
        break
        continue
    alist.append(diceroll)
print alist
```

dictionary = {}

A collection of Key, Value pairs

- Dictionaries are another collection datatype
- Use dictionaries to define an array with an arbitrary index scheme
  - Whereas lists and tuples must use integer indices
- Constructors "`dict()`" or "`{}`".
- Called "associative arrays" in other languages.

```
picnic = {}
picnic["Jerry"] = "apples"
#"Jerry" is a key
#"apples" is "Jerry"'s value
picnic["Amanda"] = "bread"
picnic["Mike"] = "salmon"
picnic["Steve"] = 3.14
print picnic["Amanda"]
print picnic["Josh"]
#he's not coming
for name in picnic:
   print name
for name in picnic:
   print name, picnic[name]
```

egSet = set()

A fast unordered (non-sequential) collection

- Keep things whose sequence doesn't matter in a set
- Fun fact! A set is backed by a hash map-- so are the keys of a dict.
    - It's decently speedier than a list to look up

```
#Do this in the console.
egSet = set()
someN = 13
#Look! 3e6 = 3000000.
while someN < 3e6:
   someN += 3
   egSet.add(someN)
#Construct a list version
egList = list(egSet)
#Which print statement takes longer?
print 2097127 in egSet
print 2097127 in egList
```

Collection Functions and Operators

Things you can do to list(), tuple(), dict() and set() objects

- `len(`some_collection`)`
  - the number of entries (integer)
- some_object `in` some_collection
  - whether a given object is in a collection (Boolean)
- some_dict`[`some_key`]`
  - the value of the some_dict at some_key
- `for` reference `in` some_collection`:` ...
  - iterates over the objects in a collection
  - at each step in the loop, the current object is accessible with “reference”
  - dictionaries return each key, not each value
- converting between collections
  - some_list `= list(`some_tuple_or_set`)`
  - some_tuple `= tuple(`some_list_or_set`)`
  - some_set `= set(`some_list_or_tuple`)`
- get a sorted list from some collection
  - some_list `= sorted(`some_collection`)`
- adding to a collection
  - some_list`.append(`some_object`)`
  - some_set`.add(`some_object`)`
  - some_dict`[`key`] =` some_value
- removing and returning a value
  - some_dict`.pop(`some_key`)`
    - returns the object at that key
  - some_list`.pop(`some_object`)`
  - some_set`.pop(`some_object`)`
- removing a value without returning it
  - some_list`.remove(`some_object`)`
  - some_set`.remove(`some_object`)`
- get the keys or values in a list from a dict
  - some_list `=` some_dict`.keys()`
  - some_list `=` some_dict`.values()`

Exceptional!

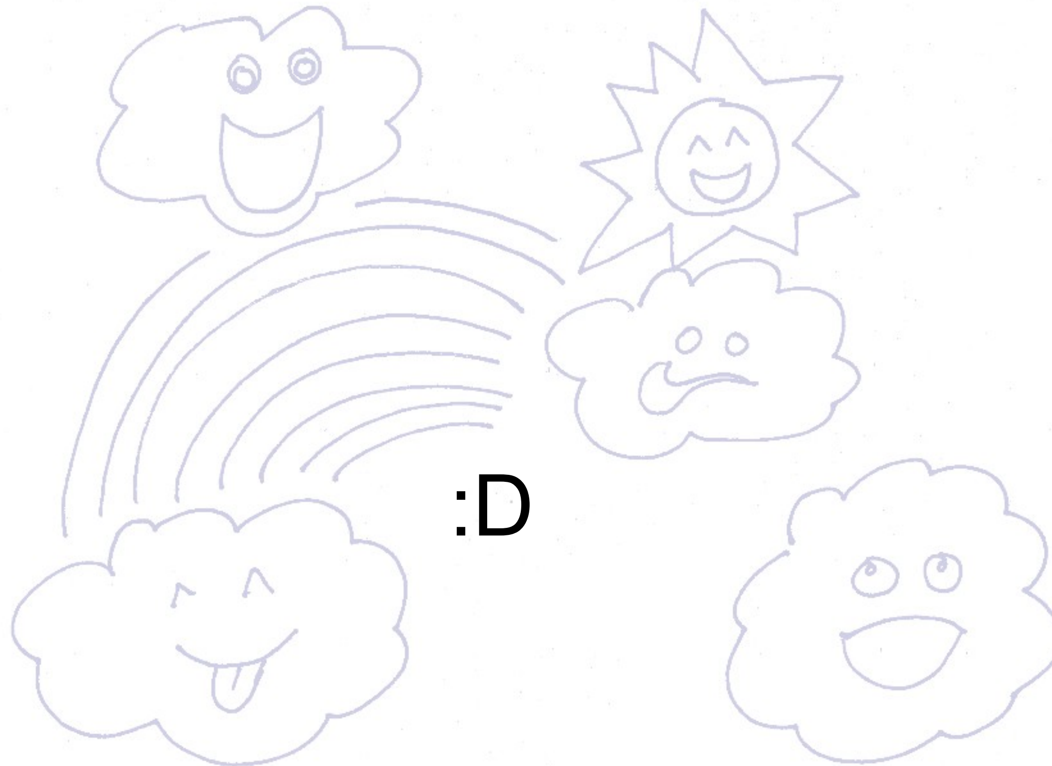
...in a way you didn't expect – Exception Handling

- Exceptions are raised when the state of your software is not expected
- Catching exceptions is optional-- you can choose to let your software crash
- Exceptional circumstances
  - Opening a file that's not there
  - Dividing by zero
  - Looking up an unassigned index in a collection
- Structure
  - `try:`
    - some block to try that may throw an exception
  - `except` someException`:`
    - stuff to do if the above exception happened
    - you can say "`except:`" to catch all exceptions
  - `else:`
    - stuff to do when no exception happens
  - `finally:`
    - stuff to do whether or not an exception happens

```
#no exception handling
def customDiv(x, y):
  z = float(x) / float(y)
  return z
print customDiv(15, 7)
print customDiv(15, 0)
#with exception handling
def customDiv(x, y):
  try:
    z = float(x) / float(y)
  except ZeroDivisionError:
    z = None
  return z
print customDiv(42, 13)
print customDiv(42, 0)
```

Half time

Next up: File handling and IO



:D

File Handling

This side up