



UC Berkeley EECS
Adj. Assistant Prof.
Dr. Gerald Friedland



Lecture #12: Quick: Exceptions and SQL

November 18th, 2016

<http://inst.eecs.berkeley.edu/~cs88>

Administrivia



- **Open Project: Starts Monday!**
 - Creative data task
 - Similar to data8, except you write the code
- **Lab Monday: SQL**
- **Lab Monday next week: Talk about Project**
- **Homework: Extra days due to Thanksgiving**
- **Lectures: This one, one more, and Q&A during RRR week**

11/18/16

UCB CS88 Fa16 L12

2

Computational Concepts Toolbox



- Data type: values, literals, operations,
- Expressions, Call expression
- Variables
- Assignment Statement
- Sequences: tuple, list
- Dictionaries
- Data structures
- Tuple assignment
- Function Definition Statement
- Conditional Statement
- Iteration: list comp, for, while
- Lambda function expr.
- Higher Order Functions
 - as Values, Args, Results
- Higher order function patterns
 - Map, Filter, Reduce
 - Function factories
- Recursion
 - Linear, Tail, Tree
- Abstract Data Types
- Mutation
- Iterators and Generators
- Object Oriented Programming
- Classes
- **Exceptions**
- **Declarative Programming**

Today: Exceptions (read 4.3)



- **Mechanism in a programming language to declare and respond to “exceptional conditions”**
 - enable non-local continuations of control
- **Often used to handle error conditions**
 - Unhandled exceptions will cause python to halt and print a stack trace
 - You already saw a non-error exception – end of iterator
- **Exceptions can be handled by the program instead**
 - `try`, `except`, `raise` statements
- **Exceptions are objects!**
 - They have classes with constructors



11/18/16

UCB CS88 Fa16 L12

3

11/18/16

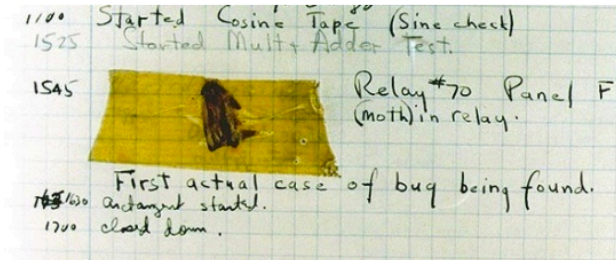
UCB CS88 Fa16 L12

4

Handling Errors



- Function receives arguments of improper type?
- Resource, e.g., file, is not available
- Network connection is lost or times out?



Grace Hopper's Notebook, 1947, Moth found in a Mark II Computer

Example exceptions



```
>>> 3/0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
>>> str.lower(1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: descriptor 'lower' requires a 'str' object
but received a 'int'
>>> ""[2]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
>>>
```

- Unhandled, thrown back to the top level interpreter
- Or halt the Python program

Functions



- Q: What is a function supposed to do?
- A: One thing well
- Q: What should it do when it is passed arguments that don't make sense?

```
>>> def divides(x, y):
...     return y%x == 0
...
>>> divides(0, 5)
???
```

```
>>> def get(data, selector):
...     return data[selector]
...
>>> get({'a': 34, 'cat': '9 lives'}, 'dog')
????
```

Exceptional exit from functions



```
>>> def divides(x, y):
...     return y%x == 0
...
>>> divides(0, 5)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in divides
ZeroDivisionError: integer division or modulo by zero
>>> def get(data, selector):
...     return data[selector]
...
>>> get({'a': 34, 'cat': '9 lives'}, 'dog')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in get
KeyError: 'dog'
>>>
```

- Function doesn't "return" but instead execution is thrown out of the function

Continue out of multiple calls deep



```
def divides(x, y):
    return y%x == 0
def divides24(x):
    return divides(x,24)
divides24(0)
```

```
ZeroDivisionError                                Traceback (most recent call last)
<ipython-input-14-ad26ce8ae76a> in <module>()
      3 def divides24(x):
      4     return divides(x,24)
----> 5 divides24(0)

<ipython-input-14-ad26ce8ae76a> in divides24(x)
      2     return y%x == 0
      3 def divides24(x):
----> 4     return divides(x,24)
      5 divides24(0)

<ipython-input-14-ad26ce8ae76a> in divides(x, y)
      1 def divides(x, y):
----> 2     return y%x == 0
      3 def divides24(x):
      4     return divides(x,24)
      5 divides24(0)
```

Python 3.3

```
1 def divides(x, y):
2     return y%x == 0
3 def divides24(x):
4     return divides(x,24)
5 divides24(0)
```

Frames

- Global frame: function divides(x, y)
- divides: function divides24(x)
- divides24: function divides24(x)
- divides24: x 0, y 24

Integer division or modulo by zero

ZeroDivisionError: integer division or modulo by zero

- Recursion/Stack unwinds until exception is handled or top

11/18/16

UCB CS88 Fa16 L12

9

11/18/16

UCB CS88 Fa16 L12

10

Flow of control stops at the exception



- And is 'thrown back' to wherever it is caught

```
def divides24(x):
    return noisy_divides(x,24)
```

```
divides24(0)
```

```
ZeroDivisionError                                Traceback (most recent call last)
<ipython-input-24-ea94e81be222> in <module>()
----> 1 divides24(0)

<ipython-input-23-c56bc11b3032> in divides24(x)
      1 def divides24(x):
----> 2     return noisy_divides(x,24)

<ipython-input-20-df96adb0c18a> in noisy_divides(x, y)
      1 def noisy_divides(x, y):
----> 2     result = (y % x == 0)
      3     if result:
      4         print("{0} divides {1}".format(x, y))
      5     else:
```

ZeroDivisionError: integer division or modulo by zero

11/18/16

UCB CS88 Fa16 L12

11

11/18/16

UCB CS88 Fa16 L12

12

Types of exceptions



- **TypeError** -- A function was passed the wrong number/type of argument
- **NameError** -- A name wasn't found
- **KeyError** -- A key wasn't found in a dictionary
- **RuntimeError** -- Catch-all for troubles during interpretation
- ...

Assert Statements



- Allow you to make assertions about assumptions that your code relies on
 - Use them liberally!
 - Incoming data is dirty till you've washed it

```
assert <assertion expression>, <string for failed>
```

- Raise an exception of type **AssertionError**
- Ignored in optimize flag: **python3 -O ...**
 - Governed by bool `__debug__`

```
def divides(x, y):
    assert x != 0, "Denominator must be non-zero"
    return y%x == 0
```

Handling Errors – try / except



- Wrap your code in `try` – `except` statements

```
try:
    <try suite>
except <exception class> as <name>:
    <except suite>
... # continue here if <try suite> succeeds w/o exception
```

- Execution rule
 - `<try suite>` is executed first
 - If during this an exception is raised and not handled otherwise
 - And if the exception inherits from `<exception class>`
 - Then `<except suite>` is executed with `<name>` bound to the exception
- Control jumps to the `except` suite of the most recent `try` that handles the exception

11/18/16

UCB CS88 Fa16 L12

13

Raise statement



- Exception are raised with a `raise` statement

```
raise <exception>
```

- `<expression>` must evaluate to a subclass of `BaseException` or an instance of one
- Exceptions are constructed like any other object

```
TypeError('Bad argument')
```

11/18/16

UCB CS88 Fa16 L12

14

Exceptions are Classes



```
class NoisyException(Exception):
    def __init__(self, stuff):
        print("Bad stuff happened", stuff)
```

```
try:
    return fun(x)
except:
    raise NoisyException((fun, x))
```

4/18/16

UCB CS88 Sp16 L11

15

11/18/16

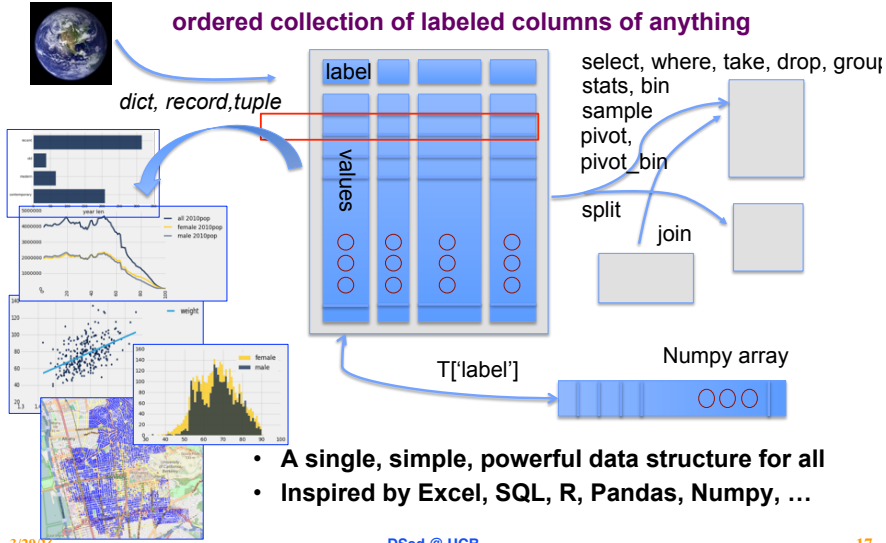
Part II – Intro to Declarative Programming SQL



UCB CS88 Fa16 L12

16

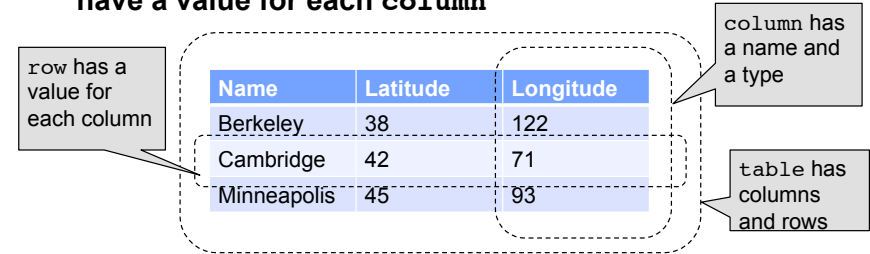
Data 8 Tables



Database Management Systems



- DBMS are persistent tables with powerful relational operators
 - Important, heavily used, interesting !
- A table is a collection of records, which are rows that have a value for each column



- Structure Query Language (SQL) is a declarative programming language describing operations on tables

SQL



- A declarative language
 - Described *what* to compute
 - Imperative languages, like python, describe *how* to compute it
 - Query processor (interpreter) chooses which of many equivalent query plans to execute to perform the SQL statements
- ANSI and ISO standard, but many variants
- **select** statement creates a new table, either from scratch or by projecting a table
- **create table** statement gives a global name to a table
- Lots of other statements
 - analyze, delete, explain, insert, replace, update, ...
- The action is in select

SQL example



- SQL statements create tables
 - Give it a try with sqlite3 or <http://kripken.github.io/sql.js/GUI/>
 - Each statement ends with ‘;’

```
culler$ sqlite3
SQLite version 3.9.2 2015-11-02 18:31:45
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> select 38 as latitude, 122 as longitude, "Berkeley" as
name;
38|122|Berkeley
sqlite>
```

select



- Comma-separated list of *column descriptions*
- Column description is an expression, optionally followed by **as** and a column name

```
select [expression] as [name], [expression] as [name]; ...
```

- Selecting literals creates a one-row table
- **union** of select statements is a table containing the union of the rows

```
select 38 as latitude, 122 as longitude, "Berkeley" as name union
select 42,          71,          "Cambridge" union
select 45,          93,          "Minneapolis";
```

Latitude	Longitude	Name
38	122	Berkeley
42	71	Cambridge
45	93	Minneapolis

11/18/16

UCB CS88 Fa16 L12

21

SQL: creating a named table



```
create table cities as
select 38 as latitude, 122 as longitude, "Berkeley" as name union
select 42,          71,          "Cambridge" union
select 45,          93,          "Minneapolis";
```

cities:

Latitude	Longitude	Name
38	122	Berkeley
42	71	Cambridge
45	93	Minneapolis

11/18/16

UCB CS88 Fa16 L12

22

create table



- SQL often used interactively
 - Result of select displayed to the user, but not stored
- Create table statement gives the result a name
 - Like a variable, but for a permanent object

```
create table [name] as [select statement];
```

SQL: using named tables - from



```
create table cities as
select 38 as latitude, 122 as longitude, "Berkeley" as name union
select 42,          71,          "Cambridge" union
select 45,          93,          "Minneapolis";
```

```
select "west coast" as region, name from cities where longitude
>= 115 union
select "other", name from cities where longitude < 115
```

cities:

Region	Name
west coast	Berkeley
other	Cambridge
other	Minneapolis

Latitude	Longitude	Name
38	122	Berkeley
42	71	Cambridge
45	93	Minneapolis

11/18/16

UCB CS88 Fa16 L12

23

11/18/16

UCB CS88 Fa16 L12

24

Projecting existing tables



- Input table specified by **from** clause
- Subset of rows selected using a **where** clause
- Ordering of the selected rows declared using an **order by** clause

```
select [columns] from [table] where [condition] order by [order];
```

```
select * from cities where longitude > 115 order by name;
```

Name	Latitude	Longitude
Cambridge	42	71
Minneapolis	45	93

Joining tables



- Two tables are joined by a comma to yield all combinations of a row from each

```
create table cities as
select 38 as latitude, 122 as longitude, "Berkeley" as name union
select 42, 71, "Cambridge" union
select 45, 93, "Minneapolis";
```

```
create table climates as
select "Berkeley" as city, "warm" as climate union
select "Cambridge" as city, "cold" as climate;
```

```
select * from cities, climates
```

latitude	longitude	name	city	climate
38	122	Berkeley	Berkeley	warm
38	122	Berkeley	Cambridge	cold
42	71	Cambridge	Berkeley	warm
42	71	Cambridge	Cambridge	cold
45	93	Minneapolis	Berkeley	warm
45	93	Minneapolis	Cambridge	cold

11/18/16

UCB CS88 Fa16 L12

25

11/18/16

UCB CS88 Fa16 L12

26

Join / Where



```
create table cities as
select 38 as latitude, 122 as longitude, "Berkeley" as name union
select 42, 71, "Cambridge" union
select 45, 93, "Minneapolis";
```

```
create table climates as
select "Berkeley" as city, "warm" as climate union
select "Cambridge" as city, "cold" as climate;
```

```
select name, climate, latitude, longitude from cities, climates
where name = city;
```

name	climate	latitude	longitude
Berkeley	warm	38	122
Cambridge	cold	42	71

Aggregation and grouping



- Reduction operators can be applied over groupings of rows

```
create table cities as
select 38 as latitude, 122 as longitude, "Berkeley" as name union
select 42, 71, "Cambridge" union
select 45, 93, "Minneapolis";
```

```
create table climates as
select "Berkeley" as city, "warm" as climate union
select "Cambridge" as city, "cold" as climate union
select "Minneapolis" as city, "cold" as climate;
```

```
select climate, min(latitude) from cities, climates where name =
city group by climate;
```

climate	min(latitude)
cold	42
warm	38

11/18/16

UCB CS88 Fa16 L12

27

11/18/16

UCB CS88 Fa16 L12

28

Summary



- **Exceptions provide a way to handle unexpected cases and errors**
- **Transfers control to enclosing handler of matching type**
 - `assert, raise <expression> , try: ... except <type> as <name>`
- **SQL a declarative programming language on relational tables**
 - largely familiar to you from data8
 - `create, select, where, order, group by, join`
- **More in lab!**