



# Exceptions & a Taste of Declarative Programming in SQL

David E. Culler

CS8 – Computational Structures in Data Science

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

Lecture 11

April 18, 2016

## 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
- Object Oriented Programming
- Classes
- Iterators and Generators



4/18/16

UCB CS88 Sp16 L11

2

## Today: Exception (read 4.3)

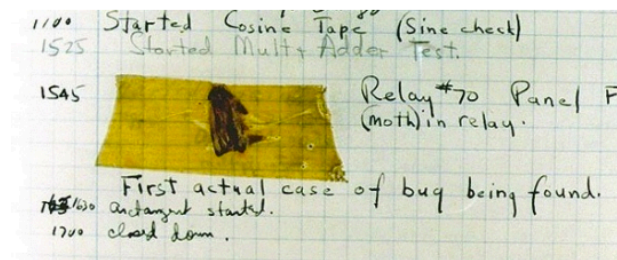


- Mechanism in a programming language to declare and respond to “exceptional conditions”
  - enable non-local continuations of control
- Often used to handle error conditions
  - Unhandled exceptions will cause python to halt and print a stack trace
  - You already saw a non-error exception – end of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

## 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

4/18/16

UCB CS88 Sp16 L11

5

## 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')

????
```

4/18/16

UCB CS88 Sp16 L11

6

## 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

4/18/16

UCB CS88 Sp16 L11

7

## Continue out of multiple calls deep

The screenshot shows a Python 3.3 interpreter window. On the left, a traceback for a `ZeroDivisionError` is displayed, showing the call stack from the top-level module down to the `divides` function. On the right, a 'Frames' panel shows the current state of the call stack. The top frame is the 'Global frame' containing the `divides` function. Below it, the `divides24` function is shown, which is currently executing the `divides(x, 24)` call. The `divides` function is also shown, with the `x` variable set to 0. The 'Objects' panel on the far right shows the objects created during the execution, including the `function divides(x, y)` object and the `function divides24(x)` object. The bottom of the window shows the current step in the traceback, which is the `ZeroDivisionError: integer division or modulo by zero`.

- Stack unwinds until exception is handled or top

4/18/16

UCB CS88 Sp16 L11

8

## 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
- ...

4/18/16

UCB CS88 Sp16 L11

9

## 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
```

4/18/16

UCB CS88 Sp16 L11

11

## 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
```

4/18/16

UCB CS88 Sp16 L11

10

## 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

4/18/16

UCB CS88 Sp16 L11

4/18/16

12

## Demo



## 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')
```

## Exceptions are Classes



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

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

## Demo





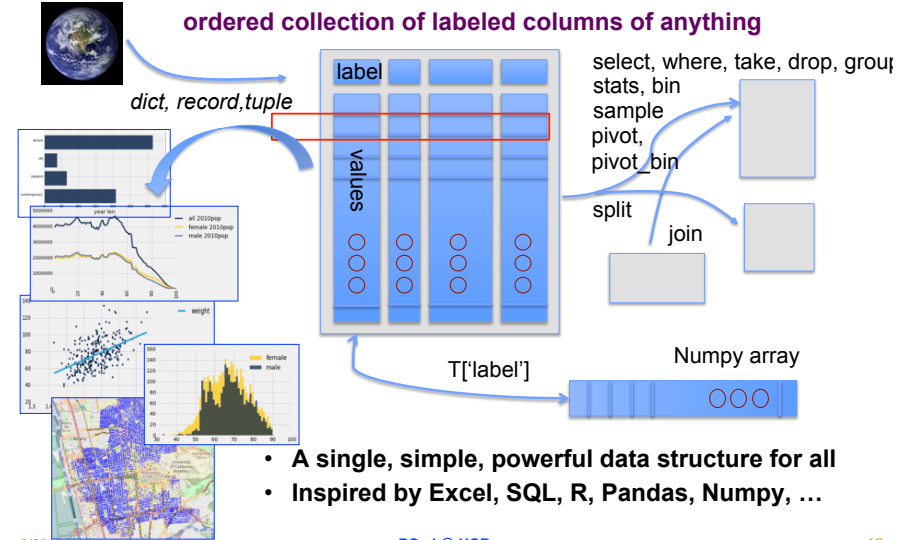
## Part II – Intro to Declarative Programming

4/18/16

UCB CS88 Sp16 L11

17

## Data 8 Tables



3/29/16

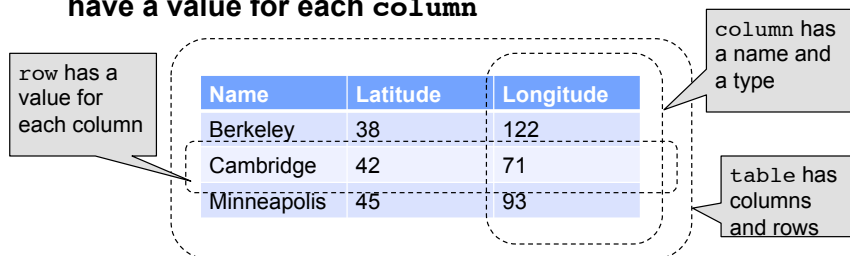
DSed @ UCB

18

## 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

4/18/16

UCB CS88 Sp16 L11

19

## 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

4/18/16

UCB CS88 Sp16 L11

20

## 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>
```

4/18/16

UCB CS88 Sp16 L11

21

## 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

4/18/16

UCB CS88 Sp16 L11

22

## 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

4/18/16

UCB CS88 Sp16 L11

23

## 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];
```

4/18/16

UCB CS88 Sp16 L11

24

## 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
```

Region	Name
west coast	Berkeley
other	Cambridge
other	Minneapolis

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

4/18/16

UCB CS88 Sp16 L11

25

## 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

4/18/16

UCB CS88 Sp16 L11

26

## 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

4/18/16

UCB CS88 Sp16 L11

27

## 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

4/18/16

UCB CS88 Sp16 L11

28

## 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

## 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 today