

UC Berkeley EECS  
Lecturer  
Michael Ball

# Computational Structures in Data Science

---



## Wrap Up: Why Linked Lists?



## Why are linked lists useful?

---

- Honestly, a list() is easier *most* of the time
  - Python handles all the hard details!
  - When data gets large, there are lots of edge cases.
- In terms of *efficiency*: Linked lists make it fast to move items around, inserts and deletes.
  - But they are slower to finding any single item.
- Linked Lists are a simplified forms of other more complicated structures
  - a Tree: Each "link" can have multiple children
  - a Graph: Each "node" can have many neighbors



## Uses for a Linked List

---

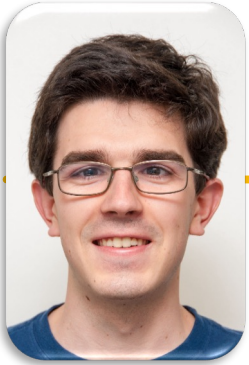
- Modeling a Polynomial Equation
  - each item is (coefficient, exponent, next\_term)
- Items in a music playlist
  - each item is a (song, next\_song) pair
  - easy to add/remove items
    - » Specifically: often want to remove the first item
- Model real-world relationships
  - Anything that is a “chain” is a good option
  - Next week: We’ll extend this idea to “trees”



## Efficiency of Linked Lists vs Lists

---

- Linked Lists generally use less memory.
- Linked Lists:
  - Once you've found an item, inserting / removing is easy,  $O(1)$
  - Finding anything other than the first/last item is  $O(n)$
- "Regular" Lists:
  - Inserting / Removing items, other than the last is  $O(n)$  – due to internal copying
  - Finding any random item is  $O(1)$ .
- What if you need to iterate over all items in order?
  - $O(n)$  in both cases



UC Berkeley EECS  
Lecturer  
Michael Ball

# Computational Structures in Data Science

---



## Lecture: Exceptions



## Learning Objectives

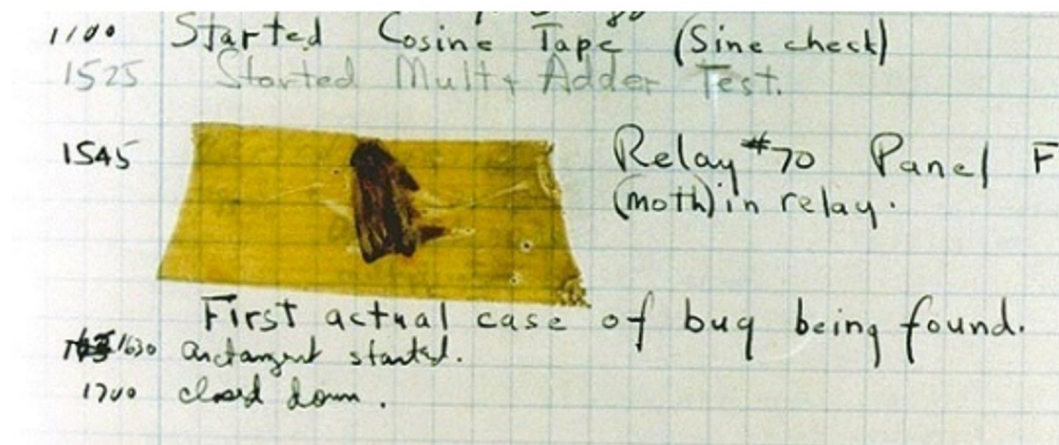
---

- Exceptions give us a formal way to address error conditions
- "Catch" exceptions in a Python Program
- Define and Raise our own exceptions



## Errors Can Occur Just About Anywhere!

- Function receives arguments of improper type?
- Resources (e.g. files or some data) are not available
- Network connection is lost or times out?



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



## Example exceptions (Docs)

```
>>> 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
>>>
```

notebook  
k

- 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



## Reading A "Stack Trace" or "Traceback" ([Docs](#))

---

- All errors in Python *should* return some structured feedback.
- Errors may be dense, but it contains some really helpful information!

👉 `python3 -i 20-Exceptions.py`

What is your age? 5

Catching CS88Error

**Traceback** (most recent call last):

```
File "...Exceptions.py", line 24, in <module>
```

```
    get_age_in_days()
```

```
File "...", line 20, in get_age_in_days
```

```
    raise e
```

```
File "...", line 14, in get_age_in_days
```

```
    raise CS88Error('You seem young!')
```

```
__main__.CS88Error: You seem young!
```



# 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)

<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)

ZeroDivisionError: integer division or modulo by zero

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)
```

[Edit code](#)

< Back Step 8 of 11 Forward > Last >>

integer division or modulo by zero

Frames

Global frame

divides

divides24

Objects

function divides(x, y)

function divides24(x)

divides24

x 0

divides

x 0

y 24

- Stack unwinds until exception is handled or top



## 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
- Your own exceptions!

# Demo

---





## 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
```



## Assert Statements

---

- Allow you to make assertions about assumptions that your code relies on
  - Use them liberally!
  - Incoming data is “dirty” and unsafe till you’ve “cleaned” 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

# Demo



```
def safe_apply_fun(f,x):  
    try:  
        return f(x)           # normal execution, return the result  
    except Exception as e:    # exceptions are objects of class deri  
        return e              # value returned on exception
```

```
def divides(x, y):  
    assert x != 0, "Bad argument to divides - denominator should be non-zero"  
    if (type(x) != int or type(y) != int):  
        raise TypeError("divides only takes integers")  
    return y%x == 0
```



## 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)
```

```
class CS88Error(Exception):  
    pass # The one time you can skip init. ;)
```

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

# Demo

---





## Summary

---

- Approach use of exceptions as a design problem
  - Meaningful behavior => methods [& attributes]
  - ADT methodology
  - What's private and hidden? vs What's public?
- Use it to streamline development
- Anticipate exceptional cases and unforeseen problems
  - try ... catch
  - raise / assert