

# LECTURE 18

## EXCEPTIONS

MCS 260 Fall 2021

Emily Dumas

# REMINDERS

- Work on Project 2, due 6pm Fri Oct 8
- Project 2 autograder now open!

# ERROR HANDLING

Programs sometimes encounter unexpected events:

- Data has unexpected format
- File operation impossible (missing, permissions, ...)
- Variable name does not exist
- *...many more*

Making a program **robust** means ensuring it can serve its function even after certain errors occur.

# ERROR HANDLING APPROACHES

Three main approaches:

- Do nothing. Behavior when an error occurs depends on OS and language. Not good!
- Explicitly check for error at every step (often using return values), report to caller if in a function.
- Exceptions (to be explained soon)

# EXPLICIT CHECKS AT EACH STEP

Build functions that return information, and an indication of whether an error occurred.

```
retval, errcode = load_data()
if errcode != 0:
    # Some error occurred
    print("Unable to load data due to error: ",errmsg[errcode])
```

When functions call other functions, this gets complicated. Each one needs to detect and report errors to its caller.

# EXCEPTIONS

An **exception** signals that an unexpected event has occurred, and control should jump to code that is meant to handle it. We say the error "raises" an exception, and other code "catches" it.

In Python, an exception behaves a bit like `break`. Just as `break` searches for an enclosing loop, after an exception Python searches for an enclosing `try` block that will catch it.

# TRY...EXCEPT

```
try:
    # code that does something that may raise an
    # exception we want to handle
except:
    # code to start executing if an error occurs

# line that will execute after the try-except
```

# Handle input string that is not a number.

```
while True:
    s = input()
    try:
        n = float(s)
        break
    except:
        print("Please enter a number.")
print("Got a number:",n)
```



Exceptions are Python's preferred error handling mechanism.

Often described as "it is easier to ask forgiveness than permission"\* or EAFP.

The contrasting approach, of anticipating error conditions and checking beforehand, is often called "look before you leap" or LBYL.

\* A phrase popularized by US Navy admiral and computer scientist Grace Hopper.

# UNCAUGHT EXCEPTIONS

If no try...except block catches an exception, the program ends.

An error message is printed that also describes what **type of exception** occurred.

```
>>> int(input())
walrus
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'walrus'
```

# SOME BUILT-IN EXCEPTIONS

- **ValueError** - Function got the right type, but an inappropriate value  
e.g. `int("apple")`
- **IndexError** - Valid index requested, but that item does not exist  
e.g. `["a", "b"][15]`
- **KeyError** - A requested key was not found in a dictionary  
e.g. `{"a": 260, "b": 330}["autumn"]`
- **TypeError** - Invalid argument type, e.g. non-integer list index:  
e.g. `["a", "b"]["foo"]`
- **OSError** - The OS reported an error in a requested operation; includes many file-related errors (e.g. file not found, filename is a directory, permissions do not allow opening the file, ...)
- **NameError** - Reference to unknown variable.

# CATCHING SPECIFIC EXCEPTIONS

```
try:
    # code that does something that may raise an
    # exception we want to handle
except ValueError:
    # code to handle a ValueError
except OSError:
    # code to handle a OSError
except:
    # code to handle any other exception

# line that will execute after the try-except
```

# CATCHING EXCEPTION OBJECTS

```
try:
    open("foo.txt", "r")
except OSError as e:
    print("Unable to open foo.txt; the error was:\n", e)
```

Printing an exception object gives some information about the error. Some exception types carry additional data, like `OSError.filename` to get the filename of the file the error involves.

# RAISING EXCEPTIONS YOURSELF

Your functions can raise exceptions using the `raise` keyword, followed by an exception type.

```
raise ValueError("U+1F4A9 not allowed in homework")
raise TypeError("This function cannot use a complex value")
raise NotImplementedError("Loading map from JSON not working y
raise Exception("Locusts.")
```

# REFERENCES

- In *Downey*:
  - Various built-in exceptions are discussed throughout.
  - [Section 14.5](#) and [Section 15.7](#) discuss catching exceptions.
- [List of built-in exceptions](#) from Python 3 documentation.

# REVISION HISTORY

- 2021-10-04 Initial publication

