EdX 6.00x Notes

Lecture 8:

- What is an exception?
    - What happens when procedure execution hits an unexpected condition?
        - Trying to access beyond the limits of a list will raise an IndexError
            - Test =[1,2,3]
            - Test[4]
        - Trying to convert an inappropriate type will raise a TypeError
            - Int(Test)
        - Referencing a non-existing variable will raise a NameError
            - A
        - Mixing data types without appropriate coercion will raise a TypeError
            - 'a'/4
    - These are **exceptions** – exceptions to what was expected
- Ways to handle exceptions:
    - Fail silently: substitute default values, continue
        - Bad idea! User gets no indication, results may be suspect.
    - Return an "error" value
        - What value to choose? None?
        - Callers must include code to check for this special value and deal with consequences -> cascade of error values up the call tree.
    - Stop execution, signal error condtion
        - In Python: **raise an exception**
            - Example: raise Exception("descriptive string")
- Dealing with exceptions
    - Python code can provide handlers for exceptions
    - Exceptions raised by statements in body of **try** are handled by the **except** statement and execution continues with the body of the **except** statement
- Handling specific exceptions
    - Usually the handler is only meant to deal with a particular type of exception. Sometimes we need to clean up before continuing
- Types of Exceptions
    - Already seen common error types:
    - SyntaxError: Python can't parse program
    - NameError: local or global name not found
    - AttributeError: attribute reference fails
    - TypeError: operand doesn't have correct tpe
    - ValueError: operand type okay, but value is illegal

    - IOerror – IO system reports malfunction (e.g. file not found)
    - ArithmeticError – arithmetic related error
- Other extensions to **try**
    - else:
        - Body of this clause is executed when execution of associated try body completes with no exceptions
    - finally:
        - Body of this clause is always executed after try, else, and except clauses, even if they raised another error or executed a break, continue or return
        - Useful for cleanup-code that should be run matter what else happened (e.g. close file)
- Exceptions as flow of control
    - In traditional programming languages, one deals with errors by having functions return special values
    - Any other code invoking a function has to check whether 'error value" was returned
    - In Python, can just raise an exception when unable to produce a result consistent with function's specification
        - Raise exceptionName(arguments)
- NaN – Not a number
- Compare to traditional code
    - Harder to read, and thus to maintain or modify
    - Less efficient
    - Easier to think about processing on data structure abstractly, with exceptions to deal with unusual or unexpected cases
- Assertions
    - If we simply want to be sure that assumptions on state of computation are as expected, we can use an **assert** statement
    - We can't control response, but will raise an AssertionError exception if this happens
    - This is good defensive programming
- Assertions as defensive programming
    - While assertions don't allow a programmer to control response to unexpected conditions, they are a great method for ensuring that execution halts whenever an expected condition is not met
    - Typically used to check inputs to procedures, but can be used anywhere
    - Can make it easier to locate a source of a bug
- Extending use of assertions
    - While pre-conditions on inputs are valuable to check, can also apply post-conditions on outputs before proceeding to next stage
- Pros & Cons to using assertions
    - Slight loss of efficiency
    - Defensive programming:

        - By checking pre- and post-conditions on inputs and output, avoid propagating bad values
- Where to use assertions?
    - Goal is to spot bugs early, and make clear where they happened
        - Easier to debug when caught at first point of contact, instead of trying to trace down later
    - Not to be used in place of testing, but as a supplement to testing
    - Should probably rely on raising exceptions if users supplies bad data input, and use assertions for:
        - Checking types of arguments or values
        - Checking that invariants on data structures are met
        - Checking constraints on return values
        - Checking for violations of constraints on procedure (e.g. no duplicates in a list)