EdX 6.00x Notes

Lecture 4:

- Black Box Abstraction:
  - Capturing the idea of a computational method inside a function, allowing us to separate the details of how we do the computation from the use of the computation.
- Functions:
  - Give us abstraction – allow us to capture computation and treat as if it is a primitive.
    - Use by simply calling name, and providing input
    - Internal details hidden from users (*black box*)
  - Syntax:
    - def <function name> (<formal parameters>)
    - <function body>
  - Function name is any legal Python name
  - Within parenthesis are zero or more formal parameters – each is a variable name to be used inside the function's body
- Function returns:
  - Body can consist of any number of legal Python expressions.
  - Expressions are evaluated until:
    - Run out of expressions, in which case special value None is returned
    - Or until special keyword return is evaluated, in which case subsequent expression is evaluated and that value is returned as a value of function call
- Summary of function call:
  - Expressions for each parameter are evaluated, bound to formal parameter names of function.
  - Control transfers to the first expression in bod of function.
  - Body expressions executed until return keyword reached (returning value of next expression) or run out of expressions (returning None)
  - Invocation I bound to the returned value.
  - Control transfers to next piece of code.
- Docstring
  - A special type of comment that is used to document what your function is doing.
  - Typically, docstrings will explain what the function expects the type(s) of the argument(s) to be, and what the function is returning.
  - In Python, docstrings appear immediately after the def line of a function, before the body.
  - Docstrings start and end with triple quotes.
- Note:
  - Perhaps contrary to expectations, in Python it is legal to compare functions!

- Environments to understand bindings:
    - Environments are formalism for tracking bindings of variables and values
    - Assignments pair name and value in environment
    - Asking for value of name just looks up in current environment
    - Python shell is default (or global) environment
    - Definitions pair function name with details of function (*procedure object*)
- Environment pointer:
    - From a procedure object, points back to the environment in which that procedure was defined.
    - Functions when called create a new environment and do not run in the global environment.
- Observations on Functions & Scoping:
    - Each function call creates a new environment, which scopes bindings of formal parameters and values, and of local variables (those created with assignments within body)
    - Scoping often called static or lexical because scope within which variable has value is defined by extent of code boundaries.
- Procedures and Frames:
    - Each call to each procedure creates its own frame.
    - It inherits from the environment where the procedure said it should.
    - You can have the same variable names in different procedures.
- Specifications:
    - Are a contact between implementer of function and user
        - Assumptions: Conditions that must be met by users of function. Typically constraints on parameters, such as type, and sometimes acceptable ranges of values.
        - Guarantees: Conditions that must be met by the function, provided that it has been called in a way that satisfies assumptions.
- Functions close the loop:
    - Can now create new procedures and treat as if Python primitives.
    - Properties:
        - Decomposition: Break problems into modules that are self-contained, and can be reused in other settings.
        - Abstraction: Hide details. Users need not know interior details, can just use as if a black box.
- Using functions in modules:
    - Modularity suggests grouping functions together that share a common theme.
    - Place in a single.py file.
    - Use import command to access.