Looping, Enumeration, Functions

6.100 LECTURE 3

SPRING 2026

Announcements

- Remember to install Python and VS Code!
  - help available during office hours **5–9 pm MTWRF**
  - Room **38-370**
- Instructor office hours
  - **Thursdays 2:30–3:30 pm**
  - Room **38-648**
- Recitation starts Friday
  - check your schedule
    - **https://registrar.mit.edu/registration-academics/registration-information/understanding-your-schedule**
  - if need to switch or get assigned a section, email **6.100-staff@mit.edu**

Muddy cards

does round(), abs(), min(), max() create new objects? (in memory)
does int() create a new object or just change the type of an object?

Looping Mechanisms

Looping with `while`

- General mechanism in Python
  - **`while`** *`condition`*`:`

    ```
        body code of loop
        ...
    ```

- Loop exits only once *`condition`* is `False`
  - *`condition`* is an unchanging expression in code
  - but its evaluation depends on what variables it references
  - so **body code** needs to **update relevant variables**

Looping with `for`

- Python syntax
  - ```
    for var in iterable:
        Loop body code
        ...
    ```
- A **Python iterable** is a certain type of object
  - produces one value at a time specifically when "queried" by the `for` mechanism
  - so far, we've encountered `str` and `range` types
  - https://docs.python.org/3/library/stdtypes.html#range
- Loop automatically exits when *`iterable`* **is exhausted**
  - `for` makes repeated assignments to variable *`var`* until then
    ```

Interrupting loop execution

- `break`
  - immediately jumps out of loop
- `continue`
  - stops current loop iteration
  - jumps back to the loop condition to start next iteration
- These work with both **`while`** and **`for`** loops
- Distinct from `pass`
  - not related to looping
  - simply executes with no effect, i.e. **no operation,** i.e. **no-op,** i.e. **noop**

Enumeration Strategies

Exhaustive enumeration

- A broad computational theme, naturally expressed with loops
- **Identify problem constraints**
  - express properties of a valid solution
  - typically identify variables and relationships between them
- **Enumerate potential solutions**
  - systematically step through solution space
- **Test each one against constraints**
  - **feasibility:** stop when find any solution
  - **optimality:** track best solution found so far
- Other names
  - **guess-and-check**
  - **generate-and-test**
  - **brute force**

Example: simple algebra problem

- **Alyssa, Ben,** and **Cindy** are selling tickets to a fundraiser.
  - Ben sells 20 fewer than Alyssa
  - Cindy sells twice as many as Alyssa
  - 1000 total tickets were sold by the three people
- **How many tickets did each sell?**
  - could solve this algebraically
  - let's try exhaustive enumeration and testing each candidate solution
- **Initial strategy**
  - solution space: each person could sell anywhere between 0 and 1000 tickets
- **More efficient strategy**
  - Ben's and Cindy's ticket counts are directly related to Alyssa's
  - directly assign their counts from Alyssa's, rather than check constraints

Example extended

- Add **Derek** to the crew
  - Derek sells more tickets than Ben
  - 4 variables, 3 equalities, 1 inequality: potentially many solutions
- Find solution that **maximizes how many tickets Derek sells**
- **Initial strategy**
  - enumerate Derek's possible counts in an inner loop
  - track best so far
- **More efficient strategy**
  - enumerate Derek's possible counts in an **outer loop in decreasing order**
  - break on first solution

Functions

Functions as contained programs

- **Defining a function**
  - accept input through **parameter** variables
  - produce output through a `return` statement
  - body code is indented
    - hence need `pass` if empty

- **Calling a function**
  - **syntax: function name** followed by **( )**
  - **argument** objects go inside parentheses
  - function body runs with parameters bound to arguments
  - **function call evaluates to object returned by body code**

Example: number of dates in a month

- Can modify our bank account code to use the **correct number of days for each month**
  - but giant `if-elif-elif-...` block in middle of `for`**-loop** is unwieldy
- Separate it out into a function call `get_num_dates(month)`
- Easier to focus on that code and express it better, too

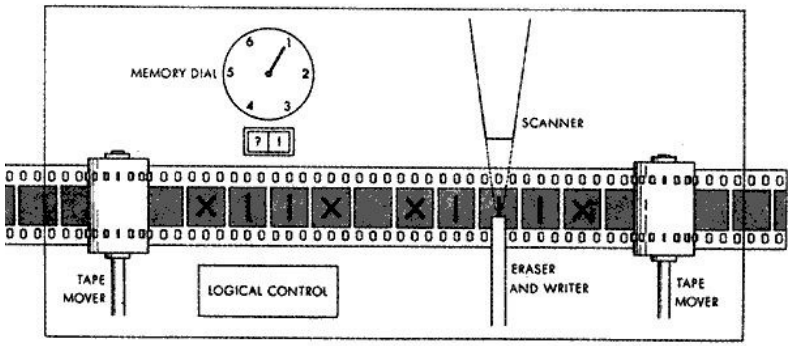
- **Takeaway**
  - **Decomposition** and **Abstraction**
  - break the original task into a sequence of smaller tasks
  - isolate subtasks with lower-level details into well-named functions

Functions going forward

- Think of functions as **small programs**
  - input through **parameters**
  - output through **return**
- Function bodies can call other functions
  - **nested function calls** during execution
- **Recommended practice:** put as much code as possible into functions
  - **function bodies only run when functions are called,** not when they are defined
  - so reduce amount of "commenting-out" needed by only **commenting out function calls**
  - **bonus:** retain **syntax highlighting**

Where we are

- Have all content needed to complete **Pset 1**
  - due next Friday 2/13
  - checkoffs during 2/16–2/20
- Have all content needed to write any possible program
  - have a Turing-complete mechanism
  - Turing machine
    - infinite tape
    - internal state
    - read/write head



- All remaining classes are canceled
  - See you at each of our three exams!

Next time

- More about how functions work
  - details of keeping track of variables
- `list` object type
  - ordered sequences of other objects
- Pre-lecture code will be posted by noon on Sunday 2/15