LECTURE 21

DISPATCH TABLES

MCS 260 Fall 2021

David Dumas

REMINDERS

- Project 2 solution will be posted Friday
- Project 3 to be announced next week; due Nov 5
- Homework 7 due tomorrow

COMMON SITUATION

Chain of of/elif/elif/else checking the same variable, taking a different action for each possible value.

```
if s == "exit":
    exit()
elif s == "help":
    print(HELP_MSG)
elif s == "next":
    x = f(x)
    print(x)
```

If we put the body of each if/elif into a function, this would look like:

```
if s == "exit":
    exit()
elif s == "help":
    do_help()
elif s == "next":
    do_next()
```

This is ok, but the similarity of all the elif blocks is suspicious. Is there a shorter way?

We can reduce duplication by storing the mapping from values to functions in a dict.

```
handlers = {
    "exit": exit,
    "help": do_help,
    "next": do_next
}

if s in handlers:
    handlers[s]() # replaces all the if/elif bodies
```

The dictionary `handlers` is an example of a **dispatch table**.

DISPATCH TABLES

A mapping from values to actions, so looking up the value associated to a key and calling it replaces a long chain of if/elif.

Advantages:

- Possible actions are stored in an actual data structure, rather than implicitly described by code.
- Make introspection possible (program can list, examing, modify the table)
- Late extensibility: Program doesn't necessarily need to know the entire table when it starts!

TERMINAL

Let's refactor our mini-terminal to perform each command through a function, and to use a dispatch table to decide which one to call.

REFERENCES

We covered dispatch tables in detail because it provided a way to demonstrate important ideas from Lecture 20 (functions are values, variadic functions, argument unpacking) in a realistic example. Dispatch tables are not covered directly in any of the optional texts.

When we discuss object-oriented programming, we'll revisit this idea.

REVISION HISTORY

- 2021-10-10 Initial publication