CSC 220 Data Structures

Python II – various topics
Parkland College Spring 2016
20160120
Dictionaries

• Index by key to get value
  • Keys can be any ‘hashable’ (basically immutable) item
  • Most common key type we’ll see are strings
• Java HashMap, std::map
• Incredibly important, will cover inner workings after midterm.
  • (Everything after midterm is really a map)
Function parameters

- *Formal parameters*: those in function definition
- *Actual parameters*: those in function call
- Pass-by-reference (like Java)
- Positional – order matters
Keyword arguments

• Explicitly name parameters, then order doesn’t matter
• Use in def to supply default values
  • Can use instead of function overloading
Variable Scope

• When/where a variable is visible
  • Rule of thumb: use the tightest scope possible (true for any language)
  • Be wary of variable shadowing
  • Avoid globals, “god” classes

• Python will use innermost or closest definition

• Functions create a scoping level
  • All blocks, in general, create a scope level
Packing/Unpacking Return

• Can return a tuple (packing)
  • return $x^2$, $x^3$
• Can receive a tuple (unpacking)
  • $s$, $c$ = `square_and_cube`(x)
• Tuples don’t always require parentheses
Docstrings

• Documentation for code user, not developer
• Accessible from `help()`
• Must start on first line of function, method, class
• Triply quoted strings
List Comprehensions

• Compact notation for generating lists
  • More common than loops for building lists
• (Can also be used for tuples, dictionaries, etc.)
• \([f(x) \text{ for } x \text{ in } \text{some function/generator}]\)
  • \([i \text{ for } i \text{ in range}(10)]\)
  • makes list of 0 through 9
Generators

• Like a function, but with lazy evaluation
  • Values only computed as needed
  • Implies some info stays around after leaving

• `yield` keyword indicates generator

• Returns an iterable object instead of a list or value

• `range()` generates numeric sequences
  • One-by-one, as needed, saves storage
 Modules

- Alternative to C/C++ style preprocessing
- Python byte code inserted into namespace
- All Python files are modules
- Accessed with `import`
Modules II

• Import from within interpreter, code is loaded and eval’d
  • Be sure not to have any ‘bare’ code or it will get executed!

• Run from command line, will load and then be assigned __name__ of ‘__main__’

• Checking __name__ analogue of #include guards
  • Should do for any code that can be either run standalone or imported from another module
Exceptions

• Function called within a `try` block
• Error handling code inside function `raises` an exception
• Exception object is caught by `except` block
• Other uses possible, but this most common model.
• (Functions can also try to catch exceptions from functions they call, etc.)
• Will see more examples later
I/O (input/output)

• `print()` for basic text output
• `input()` for basic text input
• `open()` for opening files
  • files often read line by line, examples when we get to that
• `write()` for writing to file