Functions and Control Structures

David E. Culler
CS8 – Computational Structures in Data Science
http://inst.eecs.berkeley.edu/~cs88

Lecture 3 (there is no lecture 2)
September 10, 2018

Administrative issues

• Waitlist and Concurrent Enrollment Accepted
• Weekly Schedule
  – Monday Lecture => Read => Friday Lab => Homework (Due Th)
• Lab Assignments complete
• Culler Office Hours after class – here to BIDS 190E
  – Room in the back on the right

WIMP => Program Development

• Big Idea: Layers of Abstraction
  – The GUI look and feel is built out of files, directories, system code, etc.

Computational Concepts Toolbox

• Data type: the “kind” of value and what you can do with it
  – Integers, Floats, Booleans, Strings, [ tuples ]
• Operators
  – Arithmetic: +, -, *, /, //, %, **
  – Boolean: or, and, not
  – Comparison: <, <=, ==, !=, >=, >
  – Membership: in, is, is not
  – Conditional expression: <t_exp> if <cond> else <f_exp>
• Values
  – literals, variables, results of expression

Expressions – compute a value

Valid use of operators and values
Call expression: <fun>(<arg1>, ...)

Data Science in the News

1/25/16
UCB CS88 Fa18 L3
**Call Expressions**

- Evaluate a function on some arguments
- What would be some useful functions?

- **builtin functions**
  - [https://docs.python.org/3/library/functions.html](https://docs.python.org/3/library/functions.html)
  - `min`, `max`, `sum`
- [https://docs.python.org/3/library/](https://docs.python.org/3/library/)
- `str`
- `import math; help(math)`

---

**Defining a Function**

```
def <function name> ( <argument list> ) :
    return <expression>
```

- Generalizes an expression or set of statements to apply to lots of instances
- A lot like a mathematical function
  - maps domain to range, but can do more ...
- A function should do **one thing well**

---

**Example**

```
x = 3
y = 4 + max(17, x+6) * 0.1
z = x / y
```

```
def max (x, y) :
    return x if x > y else y
```

---

**Computational Concepts Toolbox**

- Data type
- Operators
- Values
- Expressions

- **Statements** – take an action
  - Assignment Statement
    - `<variable> = <expression>`
  - Sequence of Statements
    - `x = 3
      y = 2
      print(x+y)`
Computational Concepts today

- Good Function Definitions
- Conditional Statement
- Iteration: data-driven (list comprehension)
- Iteration: control-driven (for statement)
  - Structured
- Iteration: while statement
  - More primitive and more general

Big Idea: Software Design Patterns

How to write a good function

- Name the function to describe what it does
  - Function names should be lowercase, with words separated by underscores as necessary to improve readability
- Choose meaning parameter names
  - Variable names follow the same convention as function names
- Write the docstring to explain what it does
  - Not how it does it. What does it return?
- Write doctest to show what it should do.
  - Before you write any code
- Write the code to do it

Python Style Guide: https://www.python.org/dev/peps/pep-0008/

Example: Prime numbers

```python
def prime(n):
    if n < 2:
        return False
    return all(n % i for i in range(2, int(n ** 0.5) + 1))
```

How’s this work?

A sequence data type

- A list is an object consisting of an order sequence of values
- Its literal is [item0, item1, ...]
- In data8 you’ve seen numpy arrays
Data-driven iteration

- describe an expression to perform on each item in a sequence
- let the data dictate the control
- Called “list comprehension”

\[
\begin{array}{l}
<\text{expr with loop var}> \; \text{for} \; <\text{loop var}> \; \text{in} \; <\text{sequence expr}>
\end{array}
\]

Building Tools cont.

for statement – iteration control

- Repeat a block of statements for a structured sequence of variable bindings

\[
\begin{array}{l}
<\text{initialization statements}> \\
\text{for} \; <\text{variables}> \; \text{in} \; <\text{sequence expression}>: \\
<\text{body statements}> \\
<\text{rest of the program}>
\end{array}
\]

A very basic tool

- Initialize a variable before loop
- Update it in each iteration
- Final result on exit

Putting it together
Conditional statement

- Do some statements, conditional on a **predicate** expression

\[
\text{if } \text{<predicate>}: \\
\text{<true statements>} \\
\text{else:} \\
\text{<false statements>}
\]

Optional else clause

Getting it right

- Conditional used to handle the special case
  - Guards whether the logic applies

Beware the conditional mess

- What's wrong with this function?

Combining Concepts

- Return does not have to be at the end
  - Nesting within conditionals can simplify expression

Conditional list comprehension

- Repeat a block of statements until a **predicate** expression is satisfied

\[
\text{<initialization statements>} \\
\text{while } \text{<predicate expression>}: \\
\text{<body statements>} \\
\text{<rest of the program>}
\]
Putting even more together

- Iteration not simple linear sequence
- Accumulation of values distinct from control

Computational Concepts Toolbox

- Data type
- Operators
- Values => scalars, functions & sequences
- Expressions
  - Iteration: data-driven (list comprehension)
- Sequence of Statements
  - Assignment
  - Function Definition – with doctest
  - Return
  - Conditionals

Iteration: control-driven (for statement)
  - Structured

Iteration: while statement
  - More primitive and more general