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
Data Science in the News

2018 Safe Drinking Water Data Challenge

This series of events and community-led activities includes engagements such as National Day of Civic Hacking, online tutorials, fireside chats, and hackathons. They will all culminate in a summit and awards ceremony recognizing teams and partners that have worked hard throughout the summer to ensure every Californian has access to safe drinking water. Submissions are due by October 1 and the Summit and Award
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>, ...)

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)`
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)
Defining a Function

- 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
Calling and Returning Results

Evaluate each argument expression

Statement: ...
Statement: ... <op> fun(arg exp1, ... ) <op> ...
Statement: ...
Statement: ...

Result of return expression is the value of the call expression, Continue with rest

Pass results of each arg expression in as value of parameter variable

def fun (parameter, ... ) :
    statement: ...
    statement: ...
    return <expression>

Evaluate statements of the body using these local variables
Example

```python
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
• Sequence of Statements
  – Assignment
  – Function Definition – like assigning to the function name
  – Return
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/](https://www.python.org/dev/peps/pep-0008/)
Example: Prime numbers

```python
def prime(n):
    """Return whether n is a prime number."

    >>> prime(2)
    True
    >>> prime(3)
    True
    >>> prime(4)
    False
    """
    return "figure this out"
```

Prime number

From Wikipedia, the free encyclopedia

"Prime" redirects here. For other uses, see Prime (disambiguation).

A prime number (or a prime) is a natural number greater than 1 that cannot be formed by multiplying two smaller natural numbers. A natural number greater than 1 that is not prime is called a composite number. For example, 5 is prime because the only ways of writing it as a product, $1 \times 5$ or $5 \times 1$, involve 5 itself. However, 6 is composite because it is the product of two numbers ($2 \times 3$) that are both smaller than 6. Primes are central in number theory because of the fundamental theorem of arithmetic: every natural number greater than 1 is either a prime itself or can be factorized as a product of primes that is unique up to their order.
How’s this work?

```
(datascience)CullerMac:ideas culler$ ls
__pycache__ fun.py  lab01.py  prime1.py
(datascience)CullerMac:ideas culler$ python -m doctest prime1.py
********************************************************************************
File "/Users/culler/Classes/CS88-Fa18/ideas/prime1.py", line 4, in prime1.prime
Failed example:
    prime(2)
Expected:
    True
Got:
    'figure this out'
********************************************************************************
File "<string>", line 6, in prime1.prime
Failed example:
    prime(3)
Expected:
    True
Got:
    'figure this out'
********************************************************************************
File "<string>", line 8, in prime1.prime
Failed example:
    prime(4)
Expected:
    False
Got:
    'figure this out'
********************************************************************************
1 items had failures:
  3 of  3 in prime1.prime
***Test Failed*** 3 failures.
```

1/25/16
Building some tools

```python
def divides(number, divider):
    """Return whether divider divides number evenly."
    >>> divides(3,2)
    False
    >>> divides(4,2)
    True
    """
    return (number % divider) == 0
```
A sequence data type

- A list is an object consisting of an order sequence of values
- Its literal is \([ \text{item0, item1, ...} \]
- In data8 you’ve seen numpy arrays

```python
>>> [1, 2, 3]
[1, 2, 3]
>>> x = [1, 2, 3]
>>> import numpy as np
>>> nx = np.array(x)
>>> nx
array([1, 2, 3])
>>> nx + nx
array([2, 4, 6])
>>> x + x
array([1, 2, 3, 1, 2, 3])
>>> nx*3
array([3, 6, 9])
>>> x*3
array([3, 6, 9])
>>> []
[]
>>> ```
Data-driven iteration

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

```python
[ <expr with loop var> for <loop var> in <sequence expr> ]
```
def dividers(n):
    """Return list of whether numbers greater than 1 that divide n.
    """
    return [divides(n,i) for i in range(2,n)]
Building Tools cont.

```python
def divides(number, divider):
    """Return whether divider divides number evenly.
    >>> divides(3, 2)
    False
    >>> divides(4, 2)
    True
    """
    return (number % divider) == 0

def dividers(n):
    """Return list of whether numbers greater than 1 that divide n.
    >>> dividers(6)
    [True, True]
    >>> dividers(9)
    [False, True, False]
    """
    return [divides(n, i) for i in range(2, (n//2)+1)]
```

```
culler$ python -m doctest dividers.py
culler$
```

```
[(datascience)CullerMac:ideas culler$ python -i dividers.py
 >>> dividers(17)
 [False, False, False, False, False, False, False, False]
 >>>
```
for statement – iteration control

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

<initialization statements>
for <variables> in <sequence expression>:
  <body statements>
<rest of the program>
A very basic tool

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

```python
def divides(number, divider):
    """Return whether divider divides number evenly.
    >>> divides(3,2)
    False
    >>> divides(4,2)
    True
    """
    return (number % divider) == 0

def dividers(n):
    """Return list of whether numbers greater than 1 that divide n.
    >>> dividers(6)
    [True, True]
    >>> dividers(9)
    [False, True, False]
    """
    return [divides(n,i) for i in range(2, (n//2)+1)]

def cum_OR(lst):
    """Return cumulative OR of entries in lst.
    >>> cum_OR([True, False])
    True
    >>> cum_OR([False, False])
    False
    """
    co = False
    for item in lst:
        co = co or item
    return co

def prime(n):
    """Return whether n is a prime number.
    >>> prime(2)
    True
    >>> prime(3)
    True
    >>> prime(4)
    False
    """
    return not cum_OR(dividers(n))
```

```bash
(c)ullerMac:ideas culler$ python -m doctest prime3.py
(c)ullerMac:ideas culler$ python -i prime3.py
>>> prime(17)
True
>>> prime(8)
False
>>> prime(1)
True
>>> prime(0)
True
>>> prime(-17)
True
>>> 
```
Conditional statement

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

```python
if <predicate>:
    <true statements>
else:
    <false statements>
```

Optional else clause
• 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

```python
def prime(n):
    """Return whether n is a prime number."
    >>> prime(2)
    True
    >>> prime(3)
    True
    >>> prime(4)
    False
    >>> prime(1)
    False
    ""
    if n < 2:
        return False
    for d in dividers(n):
        if d: return False
    return True

def primes(n):
    """Return primes up to n."
    return [i for i in range(2, n) if prime(i)]

[(datascience)CullerMac:ideas culler$ python -i prime5.py

>>> primes(10)
[2, 3, 5, 7]

>>> primes(100)
```
while statement – iteration control

• Repeat a block of statements until a predicate expression is satisfied

<initialization statements>
while <predicate expression>:
  <body statements>

<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