



UC Berkeley EECS
Lecturer
Michael Ball

Lecture #2: Programming Structures: Loops and Functions

September 16, 2019

<http://cs88.org>

Administrivia



- Everyone should be enrolled now
- iClickers: Start next week.

02/04/19

UCB CS88 Sp19 L2

2

Computational Concepts Today



- Fundamentals of Python
- Conditional Statements
- Functions
- Lists
- Iteration



02/04/19

UCB CS88 Sp19 L2

3

Data or Code? Abstraction!



Human-readable code
(programming language)

Machine-executable
instructions (byte code)

```
def add5(x):
    return x+5

def dotwrite(ast):
    nodename = getNodename()
    label=symbol.sym_name.get(int(ast[0].ast[0])
    print '%s (%s)' % (nodename, label),
    if isinstance(ast[1], str):
        if ast[1].strip():
            print '%s' % ast[1]
        else:
            print ''
    else:
        print ''
        children = []
        for n, child in enumerate(ast[1:]):
            children.append(dotwrite(child))
        print '%s' % '{ %s' % nodename,
        for name in children:
            print '%s' % name,
```

```
011001110001001101000010001101000100111101000111
0011011000000111110111000100001111111111011101
11111111100111010001101000100011000010010111001000
11100010101010011101101110010011101111111111111
110011111001100000000001011110100101100111110111
111111000001000110001111001100000001010111110
00001101001100100111101111000011110011001001011
10011100001000100101110011100001100001101011111
100100111111001101110001111000110111100011110
11011101101011101110011111001111100111100010011
111000100101110001100011100011111111111111111011
110111111100001100000101110011111000000011001100
1010000111001111111111111100000011000100011000
110011001101101111111001011110110111000000111111
1101100110001000100011111100111100100010001000
00001110110010011000011111011111111111100010011
1000110011001111010001100001011111000110001111
00111001111100111100111001101101111110010111111
1100111111111100100111111111111111111111111110000
0101101101101101111110100110101010111110100011
```

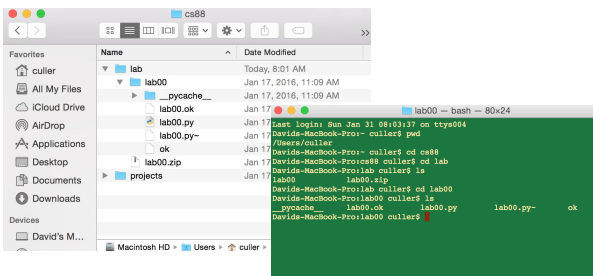
Compiler or Interpreter
Here: Python

02/04/19

UCB CS88 Sp19 L2

4

Code or GUI: More Abstraction!



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

02/04/19

UCB CS88 Sp19 L2

5

Let's talk Python



- Expression $3.1 * 2.6$
- Call expression $\text{max}(0, x)$
- Variables $x = \text{<expression>}$
- Assignment Statement `def <function name> (<argument list>) :`
- Define Function: `if ...`
- Control Statements: `for ...`
`while ...`
`list comprehension`

02/04/19

UCB CS88 Sp19 L2

6

Conditional statement

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

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

- Example:

```
if (temperature>37.2):
    print("fever!")
else:
    print("no fever")
```

02/04/19

UCB CS88 Sp19 L2

7

Defining Functions

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

return expression

- Abstracts an expression or set of statements to apply to lots of instances of the problem
- A function should *do one thing well*

02/04/19

UCB CS88 Sp19 L2

8

Functions: Calling and Returning Results

```
def my_function(number):
    print(argument)
    statements
    return number

data = my_function(1)

# data becomes whatever value is, returned.
# In this case data holds the value of number.
# 1 is an _argument_ to my_function,
# the argument number will be 1.
```

02/04/19

UCB CS88 Sp19 L2

9

Functions: 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
```

02/04/19

UCB CS88 Sp19 L2

10

How to write a good Function

- Give a descriptive name
 - Function names should be lowercase. If necessary, separate words by underscores to improve readability. Names are extremely suggestive!
- Chose meaningful parameter names
 - Again, names are extremely suggestive.
- Write the docstring to explain *what* it does
 - What does the function return? What are corner cases for parameters?
- Write doctest to show what it should do
 - Before you write the implementation.

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

02/04/19

UCB CS88 Sp19 L2

11

Example: Prime Numbers

```
1 def prime(n):
2     """Return whether n is a prime number.
3
4     >>> prime(2)
5     True
6     >>> prime(3)
7     True
8     >>> prime(4)
9     False
10    """
11
12    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, 1×5 or 5×1 , involve 5 itself. However, 6 is composite because it is the product of two numbers (2×3) 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.

Why do we have prime numbers?

<https://www.youtube.com/watch?v=e4kevnq2vPI&t=72s&index=6&list=PL17CtGMLr0Xz3vNK31TG7mJlzmF78vsFO>

02/04/19

UCB CS88 Sp19 L2

12

list – A data structure for iteration



- A list is a collection of items in a single group.
- They can hold just about anything.

```
my_list = [1, 2, 3]

my_courses = ['CS88', 'DATA8', 'MATH1A']

len(my_courses) == 3 # len returns the
length

print(my_courses[0]) # prints CS88
```

02/04/19

UCB CS88 Sp19 L2

13

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>
```

```
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
```

02/04/19

UCB CS88 Sp19 L2

14

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>
```

```
def first_primes(k):
    """ Return the first k primes.
    """
    primes = []
    num = 2
    while len(primes) < k :
        if prime(num):
            primes = primes + [num]
            num = num + 1
    return primes
```

02/04/19

UCB CS88 Sp19 L2

15

Data-driven iteration



- describe an expression to perform on each item in a sequence
- let the data dictate the control

```
[ <expr with loop var> for <loop var> in <sequence expr > ]
```

```
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) ]
```

02/04/19

UCB CS88 Sp19 L2

16