Computational Structures in Data Science

Lecture #2: Programming Structures: Loops and Functions

February 4, 2019

http://inst.eecs.berkeley.edu/~cs88

Administrivia

• If you are waitlisted: Please wait.
• If you are concurrent enrollment: Please wait.
• iClickers: Start next week.

Solutions for the Wandering Mind

A binary digit (bit) is a symbol from \( \{0, 1\} \).

• How many strings can you represent with \( N \) bits?
  Solution: \( 2^N \)
  With 0 symbols: \( 2^0 = 1 \), this is \'\'
  With 1 symbol: \( 2^1 = 2 \), this is \'0\', \'1\'
  With 2 symbols: \( 2^2 = 4 \), this is \'00\', \'01\', \'10\', \'11\'
  With 3 symbols: \( 2^3 = 8 \), this is \'000\', \'001\', \'010\', \'011\', \'100\', \'101\', \'110\', \'111\'

• Could you build a program that compresses all strings of \( N \) bits to strings of \( M \) bits (with \( M < N \)) such that you can go back to all original strings of length \( N \)? How or Why?
  Solution: No.
  \( N \) bits represent \( 2^N \) strings. Assume \( M = N - 1 \). \( M \) bits now represent \( 2^M \) strings. It is impossible to build a mapping from \( 2^N \) strings back to \( 2^M \) strings (pigeon hole principle). Example \( M=1, N=2 \): \'00\' -> \'0\', \'11\' -> \'1\' what do we do with \'01\' and \'10\'? More on this:
  https://www.youtube.com/watch?v=yZ--bbmIp_o&t=0s&index=5&list=PL17CtGMLr0Xz3vNK31TG7nJz2mfF79wF0

Computational Concepts Today

• Fundamentals: Algorithm, Code, Data, Information
• Conditional Statement
• Functions
• Iteration

Algorithm

• An algorithm (pronounced AL-go-rith-um) is a procedure or formula to solve a problem.
• An algorithm is a sequence of instructions to change the state of a system. For example: A computer’s memory, your brain (math), or the ingredients to prepare food (cooking recipe).

Think Data 8: Change or retrieve the content of a table.

Algorithm: Properties

• An algorithm is a description that can be expressed within a finite amount of space and time.
• Executing the algorithm may take infinite space and/or time, e.g. "calculate all prime numbers".
• In CS and math, we prefer to use well-defined formal languages for defining an algorithm.

\[
6 \div 2(1+2) = ? \text{ or } 9
\]
Algorithm: Well-definition

Algorithms early in life (1st grade)

Carry (MSD)
operator +
operands

Least significant digit of result

Algorithms early in life (in binary)

• Code
A sequence of symbols used for communication between systems (brains, computers, brain-to-computer)

• Data
Observations

• Information
Reduction of uncertainty in a model (measured in bits)

More Terminology (intuitive)

Data or Code?

00000000 10000000 01000001 10000000 00010000 00000000 10000001
01000001 10000001 00010000 00000000 10000002 01000001 10000002
00100001 00000000 10000003 01000001 10000003 00010000 00000000 00000000
10000001 01000000 11000000 00100000 00010000 00000000 10000001
01000001 10000001 00010000 00000000 10000002 01000001 10000002
00100001 00000000 10000003 01000001 10000003 00010000 00000000 00000000
10000001 01000000 11000000 00100000 00010000 00000000 10000001
01000001 10000001 00010000 00000000 10000002 01000001 10000002
00100001 00000000 10000003 01000001 10000003 00010000 00000000 00000000
00000000 10100000 01000001 10011111 00110000 00000000 10000001
Here is some information!

- Integer
- String

Human-readable code (programming language)

Machine-executable instructions (byte code)

Compiler or Interpreter

Here: Python

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

Let’s talk Python

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

Conditional statement

- Do some statements, conditional on a predicate expression

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

Example:

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

Defining Functions

- Abstracts an expression or set of statements to apply to lots of instances of the problem
- A function should do one thing well
Functions: Calling and Returning Results

Evaluate each argument expression
Pass results of each argument expression in value of parameter variable
Result of return expression is the value of the call expression. Continue with rest.
Evaluate statements of the body using these local variables.

Functions: Example

\[
x = 3
y = 4 + \max(17, x + 6) \times 0.1
z = x \div y
\]

\[
def \text{max} (x, y): \nonumber
\quad \text{return } x \text{ if } x > y \text{ else } y
\]

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/

Example: Prime Numbers

Why do we have prime numbers? https://www.youtube.com/watch?v=e4kevnmq2vP&index=72s&list=PL17CtGMLr0Xz3vNK31TG7n8xm7F7q7EFO

for statement – iteration control

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

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

while statement – iteration control

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

```python
def first_primes(k):
    """Return the first k primes.
    >>> first_primes(1)
    [2]
    >>> first_primes(2)
    [2, 3]
    >>> first_primes(3)
    [2, 3, 5]
    >>> first_primes(4)
    [2, 3, 5, 7]
    """
    primes = []
    num = 2
    while len(primes) < k:
        if prime(num):
            primes = primes + [num]
        num = num + 1
    return primes
```
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)]

Thoughts for the Wandering Mind

• Could we build a complete computer that has no instructions, only data?