Administrivia

- Homework 1 is posted (due Friday)

CS101: Introduction to Computing for Science and Engineering

<table>
<thead>
<tr>
<th>What</th>
<th>Where</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time/place</td>
<td>Mon/Wed 9:00am-9:50am <a href="https://courses.engr.illinois.edu/cs101/">Catalog</a></td>
</tr>
<tr>
<td>Class URL</td>
<td><a href="https://courses.engr.illinois.edu/cs101/">https://courses.engr.illinois.edu/cs101/</a></td>
</tr>
<tr>
<td>Web forum</td>
<td>Piazza »</td>
</tr>
</tbody>
</table>

Homework

<table>
<thead>
<tr>
<th>Assignment</th>
<th>Due date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Homework 1</td>
<td>Monday, August 31st</td>
</tr>
</tbody>
</table>
Administrivia

• i>clicker
  – Wednesday and Friday i>clicker tests
  – After that, i>clicker attendance
  – Need to register device **ON COURSE COMPASS PAGE**
WHY ARE WE HERE?
I’ve seen things you people wouldn’t believe...
Everyone should program!!!

- This is my **mission**!
Why learn to program?

a. Pervasive
b. Lucrative
c. Transferrable
d. Creative
e. Self-determination
Everyone can program

- Our goal is a safe, friendly, inclusive atmosphere for everyone to learn.
- You should feel **welcome**, regardless of gender, gender identity, ethnicity, nationality, religion, disability, sexual orientation, class, political views, or educational background.
- We are **peers** and **allies**. Let’s all treat one another with respect and kindness.
Programming is a **skill!**

- Learn by *doing!*
- Learn by *interacting!*
- You need to *constantly* practice.
- *Get help* when you need it!
- **WARNING**: If you are not committed to this class, you’re not going to make it!
HOW WILL THIS CLASS WORK?
Grading

• 20% homework
• 25% labs
• 10% lecture attendance (i>clicker)
• 20% midterms (2)
• 25% final exam

Official grade book will be on Compass
Required Supplies

• *No* textbook!
• i>clicker
• CodeLab account
Course website

- https://relate.cs.illinois.edu/course/SU2016CS101/
  - Homework assignments
  - Course calendar
  - Course policies

- Can also find it via CoE course directory
  - https://courses.engr.illinois.edu
Labs

• Start tomorrow
• Intended to get hands on experience
• First learn basics of Unix command line
• Then learn some programming basics
Policies

• No late homework submissions
• All machine generated grades are final
• Late registrants should keep up with work
  – No extensions or exceptions for late registration
• **Never** copy code
Getting help

• Piazza
  – Be civil to peers and staff
  – All posts containing solutions should be made private
  – A privilege, not a right

• Office hours (on RELATE)
  – Have specific questions if you want assistance
Course Overview

• ≈ 4-5 weeks: programming (Python)
• ≈ 2-3 weeks: engineering programming
• 1 week: Matlab
WHAT IS PROGRAMMING?
Program

- A set of instructions a computer executes to achieve a goal
- Can be very long (millions of instructions)
- Also called “code” or “source code”
- Our programs will be called “scripts”
Data

• Information stored in a computer is called *data*.

• All data is represented in *binary*.
  – A series of 0’s and 1’s

• Each 0 or 1 is called a *bit*.

• Bits are stored in groups of 8 called *bytes*.
Instructions

- Programs are data.
- Instructions are encoded in binary.
- Each instruction is typically 4 or 8 bytes.
Programming Language

- An artificial language used to communicate instructions to a computer
- Rigorous and unambiguous
- Grammar is mathematically formal
- Has syntax and semantics like a natural language

\[ x = y + z \]

\texttt{add \$t0, \$t1, \$t2}
Programming Languages

• Low-level: `add $t0, $t1, $t2`
  – Define individual, machine readable instructions

• High-level: `x=y+z`
  – Human readable instructions translated into machine readable instructions
High-level languages

- Compiled languages
  - Compiler translates *entire* program into machine language
- Interpreted (scripting)
  - Interpreter translates program into machine language *line by line*
  - Translation happens “on the fly”
Python

- High-level language
- Interpreted language
- Strongly, dynamically typed language

**WARNING**: Split between versions 2 and 3. We will use version 3!
Why Python?

• Freely available
• Cross platform
• Widely adopted
• Well documented
• Designed for teaching
• Beautiful
A set of instructions a computer executes to achieve a goal is called…

a) a process.
b) a program.
c) a procedure.
d) a pronoun.
A grouping of 8 bits is called…

a) a nibble.
b) a chomp.
c) a byte.
d) a gobble.
Python is...

a) a high-level language.

b) a low-level language.
Python is...

a) a compiled language.
b) an interpreted language.
BASIC PYTHON SYNTAX AND SEMANTICS
Computer
Computer

Processor
Literals

• Describe data that doesn’t change
• **ANALOGY**: Literals are *nouns* in Python
• Represent a fixed *value* (e.g. 3 or 5,136,833,998)
Computer

Processor

3
Operators

• Describe how to *manipulate* data

**ANALOGY**: Operators are the *verbs* of Python

• Common mathematical operators (e.g. +, -, *, /) are operators

• There are *many* more operators
Computer

+ 

Processor
Expressions

• Combining constants and operators, we can build *expressions*

• **ANALOGY**: Expressions are *sentence fragments* in Python

• Expressions are *evaluated* to produce a new value (e.g. 3*5 or 23−100)

• Expressions can be very complicated (e.g. 3+8*5+4−7/100)
Computer

Processor

15
Order of Operations

- 1+1*2
  a) 4
  b) 3
  c) None of the above

- Like math, Python has order of operations
- Not always intuitive
- When in doubt, use parentheses!
Evaluate this expression:
$23 + 6/2 - 4$

a) 22
b) 18
c) -9
d) None of the above
Other operators

• Modulo
  – Symbol: %
  – Description: remainder after division
  – Example: $9 \% 2$

• Exponentiation
  – Symbol: **
  – Description: base to the exponent power
  – Example: $3^{**2}$

YES, THIS IS IMPORTANT!
Evaluate this expression:
\((28 \% 5)^3\)

a) 8
b) 27
c) 64
d) None of the above
Bitwise operators

- **YOU DON’T NEED TO KNOW THESE**
- Operate on *binary* representation
- Bitwise or |
- Bitwise xor ^
- Bitwise and &
- Shift left <<
- Shift right >>
Evaluate this expression:

1^2

a) 0  

b) 1  

THIS IS WRONG!!!

c) 2  

d) 3

The Python exponentiation operator is ** not ^
Computer

Processor
Problem

• Computer is in the same *state* as when we started
• Programs are *complex*
• We computer to *remember* the results
• We need to *store* the resulting value
Problem

• How do we know where data “lives”?  
• In low-level languages, data has an “address” represented in binary

ADD DATA AT  1010110111010100 
TO DATA AT  1101010001001001 
STORE RESULT  0000110101001110 
YUCK!!!!
Solution

• Give the memory locations a "name"
Variables

• A name for a memory location used to store data
• **ANALOGY**: Variables are *nouns* in Python.
• Variables store a *value*
• The value stored in a variable *can change over time*
• A variable is a place holder
Assignment

• Stores a value in a variable (memory)
• Uses the = symbol
  – Variable on the left
  – Expression on the right
• Example: x=3 stores the value 3 in variable x
• Defines (names) the variable if we have not already used it
x = 15 + 7 * 9
What value is stored in variable x?

a) 3
b) 31
c) 55
d) 78
x = 15 + 7 * 9
x = 3

What value is stored in variable x?
a) 3
b) 31
c) 55
d) 78
x = 3 * 5

Memory

x: 15

Processor

3 * 5

15
Statement

• A statement changes the state of the computer

• **ANALOGY**: Statements are sentences in Python.

• An assignment is a statement

• Our programs will be a series of statements
Script

- A file containing a series of Python statements
- Stored in text (no magic, just text)
- Each instruction is executed top to bottom
  - Starting from the first line
- Together, the statements form a program
Example Program

\[ \begin{align*}
x &= 10 \\
y &= x**2 \\
y &= y + y
\end{align*} \]
x = 10
y = x + 1
y = x * y

What is the value of y?

a) 11
b) 100
c) 110
d) None of the above.
What do we call x?

a) a literal
b) a variable
c) an expression
d) a statement
x=10
y=x+1
y=x*y

What do we call 10?

a) a literal
b) a variable
c) an expression
d) a statement
What do we call \( y=x \cdot y \)?

a) a literal
b) a variable
c) an expression
d) a statement
DATA TYPES
Encoding

01001000010001010100110001001111

• What does this binary data represent?
• How does the processor know?
• Unless we know the encoding we cannot interpret the data.
Types

- *Types* define the encoding in Python
- All values in Python have a *type*
- Defines how data is represented in memory
- Defines the operations that are allowed and how they work
NUMERIC TYPES
Encoding Numbers

• Numeric types are represented in binary
  – Fixed-length (only a certain number of bytes)
    1: 0001  2: 0010  3: 0011  4: 0100
    5: 0101  6: 0110  7: 0111  8: 1000
    … 15: 1111
• If we add more, this causes an overflow
  – We’ve run out of bits!
• Negative numbers? Add a sign bit
Integers

• So far, this has been our only type
• Represent integers
  – Positive and negative whole numbers
• Literals are just integers (e.g. -128)
• Cannot represent all integers
  – Python scales integer sizes
  – Only integers that fit in memory
  – Bigger integers = SLOWER program
Integer operations

• Evaluating an expression of integers will generally result in an integer answer
  – 3 + 5

• EXCEPTION: DIVISION!
  – 3/4  = 0.75

• Integer division operator
  – 3 // 4  = 0
  – Floor of division
Floats

- Referenced in the homework.
- Represent real numbers
  - Anything with a decimal point
- Literals have a decimal point (e.g. 3.0)
- Cannot represent all reals
  - Some are too large/small
  - Cannot represent arbitrary precision (e.g. π)
Floating point operations

- Evaluating an expression of floats will result in a floating point answer
- Engineers and scientists will need to be careful about precision of operations
Floats with Integers

- We can use floats and integers in the same expressions.
- The resulting value is a *floating point*.
- Operations default to most general numeric type.
Complexes

• Represent numbers on complex plane
  – Numbers with an imaginary component
• Imaginary component referred to with j
  • e.g. 2+1.3j
• They’re “jimaginary” numbers!
x = 4
y = 3 + 1j
z = 33.3333

print(x + y + z)

What is printed?
a) 40
b) 40.3333
c) 40.3333 + 1j
d) None of the above
Attribute operator

• “Reaches in” to a value to access part of its data (called an attribute)
• Extracts special variables stored “inside” the type.

print(x.real)
print(x.imag)
• Both of these components are floats.
x=(3.5+1j)
y=1
z=x+y
What is the value of z.imag?
a) 4.5+1j
b) 4.5
c) 1.0j
d) 1.0
STRING TYPE
Encoding Text

• Each symbol is stored individually.
  – Each symbol is one byte long
  – Represented by ASCII code

01001000 01000101 01001100
01001100 01001111

72 69 76

76 79
## ASCII Table

<table>
<thead>
<tr>
<th>Code</th>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x05</td>
<td>ENQ</td>
<td>Enquiry</td>
</tr>
<tr>
<td>0x06</td>
<td>ACK</td>
<td>Acknowledge</td>
</tr>
<tr>
<td>0x07</td>
<td>BELL</td>
<td>Bell</td>
</tr>
<tr>
<td>0x08</td>
<td>BS</td>
<td>Backspace</td>
</tr>
<tr>
<td>0x09</td>
<td>TAB</td>
<td>Horizontal tab</td>
</tr>
<tr>
<td>0x0A</td>
<td>LF</td>
<td>New line</td>
</tr>
<tr>
<td>0x0B</td>
<td>VT</td>
<td>Vertical tab</td>
</tr>
<tr>
<td>0x0C</td>
<td>FF</td>
<td>Form Feed</td>
</tr>
<tr>
<td>0x0D</td>
<td>CR</td>
<td>Carriage return</td>
</tr>
<tr>
<td>0x0E</td>
<td>SO</td>
<td>Shift out</td>
</tr>
<tr>
<td>0x0F</td>
<td>SI</td>
<td>Shift in</td>
</tr>
<tr>
<td>0x10</td>
<td>DLE</td>
<td>Data link escape</td>
</tr>
<tr>
<td>0x11</td>
<td>DC1</td>
<td>Device control 1</td>
</tr>
<tr>
<td>0x12</td>
<td>DC2</td>
<td>Device control 2</td>
</tr>
<tr>
<td>0x13</td>
<td>DC3</td>
<td>Device control 3</td>
</tr>
<tr>
<td>0x14</td>
<td>DC4</td>
<td>Device control 4</td>
</tr>
<tr>
<td>0x15</td>
<td>NAK</td>
<td>Negative ack</td>
</tr>
<tr>
<td>0x16</td>
<td>SYN</td>
<td>Synchronous idle</td>
</tr>
<tr>
<td>0x17</td>
<td>ETB</td>
<td>End transmission block</td>
</tr>
<tr>
<td>0x18</td>
<td>CAN</td>
<td>Cancel</td>
</tr>
<tr>
<td>0x19</td>
<td>EM</td>
<td>End of medium</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code</th>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x25</td>
<td>%</td>
<td></td>
</tr>
<tr>
<td>0x26</td>
<td>&amp;</td>
<td></td>
</tr>
<tr>
<td>0x27</td>
<td>'</td>
<td></td>
</tr>
<tr>
<td>0x28</td>
<td>(</td>
<td></td>
</tr>
<tr>
<td>0x29</td>
<td>)</td>
<td></td>
</tr>
<tr>
<td>0x2A</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>0x2B</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>0x2C</td>
<td>,</td>
<td></td>
</tr>
<tr>
<td>0x2D</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>0x2E</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>0x2F</td>
<td>/</td>
<td></td>
</tr>
<tr>
<td>0x30</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0x31</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0x32</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>0x33</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>0x34</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>0x35</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>0x36</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>0x37</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>0x38</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>0x39</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>0x40</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>0x41</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>0x42</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>0x43</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>0x44</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>0x45</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>0x46</td>
<td>G</td>
<td></td>
</tr>
<tr>
<td>0x47</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td>0x48</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>0x49</td>
<td>J</td>
<td></td>
</tr>
<tr>
<td>0x4A</td>
<td>K</td>
<td></td>
</tr>
<tr>
<td>0x4B</td>
<td>L</td>
<td></td>
</tr>
<tr>
<td>0x4C</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>0x4D</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>0x4E</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>0x4F</td>
<td>P</td>
<td></td>
</tr>
<tr>
<td>0x50</td>
<td>Q</td>
<td></td>
</tr>
<tr>
<td>0x51</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>0x52</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>0x53</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>0x54</td>
<td>U</td>
<td></td>
</tr>
<tr>
<td>0x55</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>0x56</td>
<td>W</td>
<td></td>
</tr>
<tr>
<td>0x57</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>0x58</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>0x59</td>
<td>Z</td>
<td></td>
</tr>
</tbody>
</table>

**Explanation:**
- **Code**: ASCII code for the character.
- **Character**: The character represented by the code.
- **Description**: A brief description of the character's purpose or usage.
Strings

• Literals: text surrounded by quotes
  – e.g. “TACO”
• Each symbol is called a character
• Unlike numeric types, strings can vary in length!
String operations

• **Concatenation**: combine two strings
  – Uses the + symbol
  – Example: “CS”+”101”

• **Repetition**: repeat a string
  – Uses the * symbol
  – Example: “HELLO! ”*10

• **Formatting**: used to encode other data as a string
  – Uses % symbol
Formatting operator

• Creates a string with a value stuck inside
  – Formatting them nicely
  – Have to indicate the type of the value INSIDE the string with a special code

\[
x = 100 \times 54
\]

\[
s = \text{“String is: } \%i\text{” } \% x
\]

print s
Example

name=“Ryan”
grade=2/3
m1=“Hello, %s!” % name
m2=“Your grade is: %f” % grade
print m1
print m2
x=3
s=("%i" % (x+1))*x**(5%x)
print(s)

What is the output of this program?

a) 333333333333
b) 4444444444
b) 9999
b) %i%i%i%i%i
Indexing operator

• Extracts a *single* character
• Use an integer surrounded by brackets
  – e.g. a[0]
  – Call integer the “index”

**WARNING**: We start counting from 0

• Can use negative numbers
  – Starts from end (e.g. -1 is the last character)
s=“ABCDE”

i=3

x=s[i]

What is the value of x?

a) A
b) B
c) C
d) D
e) E
s="ABCDE"
i=25\%3
x=s[i]

What is the value of x?
a) A 
b) B 
c) C 
d) D 
e) E
s="ABCDE"
i=(11\%3)−7
x=s[i]
What is the value of x?
a) A  
b) B  
c) C  
d) D  
e) E
x="3"
y=10\%4
print(x*y)

What is the output of this program?

a) 2
b) 104
c) 33
d) 3104
c=(10+5j)
i=25
r=c.real+i

What is the value and type of r?
a) An integer with value 35
b) A complex with value 35+5j
c) A float with value 35.0
d) None of the above.
Which of these expressions will cause an overflow?
a) $10^{100000}$
b) “10” * 100000
c) $10.0^{100000}$
d) None of the above
x="10"
y="%i"
print( (x+y) % 2)

What is the output of this program?
a) 102  
b) 1111  
c) 1010  
d) None of the above
STRING TYPE
Strings

- Literals: text surrounded by quotes
  - e.g. “TACO”
- Each symbol is called a **character**
- Unlike numeric types, strings can vary in length!
String operations

• **Concatenation**: combine two strings
  – Uses the + symbol
  – Example: “CS”+”101”

• **Repetition**: repeat a string
  – Uses the * symbol
  – Example: “HELLO! ”*10

• **Formatting**: used to encode other data as a string
  – Uses % symbol
Formatting operator

• Creates a string with a value stuck inside
  – Formatting them nicely
  – Have to indicate the type of the value INSIDE the string with a special code

\[ x = 100 \times 54 \]
\[ s = "String is: {\%i}" \% x \]
print s
Example

name="Ryan"
grade=0.95
m1="Hello, %s!" % name
m2="Your grade is: %f" % grade
print(m1)
print(m2)
Indexing operator

- Extracts a single character
- Use an integer surrounded by brackets
  - e.g. a[0]
  - Call integer the “index”

**WARNING**: We start counting from 0
- Can use negative numbers
  - Starts from end (e.g. -1 is the last character)
my_string="ABCDE"
i=3
x=my_string[i]
What is the value of x?
a) A
b) B
c) C
d) D
e) E
my_string=“ABCDE”
i=25%3
x=my_string[i]
What is the value of x?
a) A  
b) B  
c) C  
d) D  
e) E
my_string=“ABCDE”
i=(11%3)-7
x=my_string[i]
What is the value of x?
a) A  
b) B  
c) C  
d) D  
e) E
Slicing

• Extracts a **substring** from a string
• Similar to indexing notation
  – We can specify a **range** inside the brackets using : (colon) character
  – e.g. “Taco salad”[0:4]
• Character at first index **is included**
• Character at last index **is not included**
my_string="ABCDE"
x=my_string[1:3]

What is the value of x?

a) AB  
b) ABC  
c) BC  
d) BCD  
e) CD
FUNCTIONS
Functions

• A small program we can run within Python
  – Saves us from having to rewrite code
  – Don’t reinvent the wheel!

**ANALOGY:** Functions are *verbs* in Python.

• Also called a *subroutine* or *procedure*
Function calls

• When we want to execute a function, we **call** it or **invoke** it
• Use name of the function with parentheses
  – Example: `print()`
• Many functions are part of the Python language
  – We call them **built-in functions**
Arguments

• Functions can act on data
• Arguments are the *input* to a function
• The function *returns* a value
• Return values are the *output* of a function
• Examples:
  – print(10)
  – len(“TACO TUESDAY”)
  – abs(-123)
Arguments

• A function can take more than one argument
• Multiple arguments are separated by commas
• Examples:
  - min(1,4,5)
  - max(1,4,5)
Type conversion

• Built-in functions that convert data of one type to another

• Examples:
  – float("0.3")
  – str(3+5j)

• Some type conversions don’t work:
  – int("TACO")
  – int(3+5j)
User input

• `input()` is a built-in function
• Argument: string prompting the user
• Return value: string user typed before hitting “ENTER”
s="%"+"i"
i=3/6
x=float(s%i)*2

What is the value of x?
a) 0.0
b) "%i%i"
c) 1.0
d) "1.0"
e) None of the above.
s = "TACO TUESDAY"[2:6]
t = int(3.7)
x = s[-1] + s[t-2]

What is the value of x?

a) "0  

b) "UO"

c) "TC"

d) "TO"
s="TACO TUESDAY"[2:6] 0123456789...

\[s = \text{CO TU} \]

s="CO T"

t=int(3.7)

t=3

x=s[-1]+s[t-2]

x="T"+s[1]

x="TO"
i = len("TACO TUESDAY")
c = (1.0 + 2.0j) * (-i)
x = abs(min(c.real, -13))

What is the value of x?

a) 0
b) 11
c) 12
d) 13
Goal

- Purpose of a program is to *achieve a goal*!
- Let’s write a quadratic equation solver!
print("QUADRATIC SOLVER")
print("ax^2+bx+c=0")

a=float(input("a: "))
b=float(input("b: "))
c=float(input("c: "))

root=(b**2-4*a*c)**.5
denom=2*a
pos=(-b+root)/denom
neg=(-b-root)/denom

message1="%.2f + %.2fi" % (pos.real,pos.imag)
message2="%.2f + %.2fi" % (neg.real,neg.imag)

print("Solution 1: %s" % message1)
print("Solution 2: %s" % message2)
Methods

• Like attributes, **functions** can be stored inside the type, too.

• Use **attribute operator** on the value.
  “STOP SHOUTING!”.lower()
  (1+1j).conjugate()

Value is treated like an argument.
String methods

"GATTACA".count("A")
"MVEMJSUN".find("J")
"ABACAB".replace("AB","G")
"HAM".strip()
"clint barton".title()
"wEiRd".swapcase()
s="TACO TUESDAY"

x=s[0:s.find(" ")].lower()

x=x.title().swapcase()

What is the value of x?

a) "tACO"
b) "tuesdaY"
c) "Taco  

d) "TUESDAY"