CIS 122 Introduction to Programming and Problem Solving
Spring 2015

Course Description
CIS 122 is an introduction to computational problem solving, algorithm design, and programming in Python. This course introduces topics in computational thinking, including techniques for program design, implementation, and testing. Regular class meetings are supplemented with weekly computer labs.

This course is intended for students with no prior programming experience in any language. CIS 122 is appropriate for students from any major who are interested in a practical, one course introduction to computing. It is also appropriate for students interested in gaining programming experience prior to beginning the computer science introductory sequence for majors.

CIS 122 is in the science group, and can count toward the Bachelor of Science mathematics/computing requirement.

What is programming?
Computational problem solving;

start with a task (for example, a calculation to perform, an idea to implement, a domain to explore, a problem to solve, etc.) and apply a computational process to obtain a desired outcome:

\[
\text{TASK} \rightarrow \text{COMPUTATIONAL PROCESS} \rightarrow \text{DESIRED OUTCOME}
\]

A computational process is a sequence of well-defined operations that leads from an initial starting point to a desired final outcome.

FOR EXAMPLE:

How many students are in class today?

COMPUTATIONAL PROBLEM SOLVING:
- desired ending point/output
- initial starting point/input
- description of how to move systematically from the input to the output

\[
\text{TASK} \rightarrow \text{Computational Thinking} \rightarrow \text{COMPUTATIONAL PROCESS/ALGORITHM} \rightarrow \text{DESIRED OUTCOME}
\]

Computational thinking is the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent. [CunySnyderWing10]
A computer program implements an algorithm on a computer.

**Coding** is a process that leads from an algorithm to an executable program.

Designing, writing, testing, debugging

**Programming** = Computational Thinking + Coding

**Why (should you learn to) program?**

(1) Gain fluency in computing which is ubiquitous in our lives today.

“Program or be programmed.”
-- Rushkoff

(2) “I think everyone should know how to program a computer, because it teaches you to think. I view computer science as a liberal art, something everyone should learn to do.”
-- Steve Jobs
Programming is a fun and useful intellectual exercise which develops or enhances valuable transferable computational thinking skills, including logical, creative, design skills.

-- Campbell, et. al.

Why (should you learn to) program?

(3) Gain a tremendously powerful problem tool.

“Advances in computing have expanded our capacity to solve problems at a scale never before imagined, using strategies that have not been available to us before.”

“The best signpost to the future I know is to follow whatever happens after the word ‘computational’ ... [when] one applies ... computers to traditional domains and sees what happens.”

Why (should you learn to) program?

(4) Programming is a fundamental part of computer science for minors/majors/professionals, and provides exposure to many topics in the field of computer science.

5) CIS 122 counts toward UO B.S. math/computing requirement, UO science group requirement, and as a CIS 210 programming prerequisite.

How to Program:

Need a way to communicate with the computer:

Natural language?
Flip switches?
Programming language
How to Program:

Programming languages like Python are
- Formal
- Precise
- Unambiguous
- Readable

Why Python?
- Accessible to entry-level programmers and also for experts - like chess or tennis
- Python is widely used in many fields
- Interactive, high-level, syntax-light language - Exploring is easy, concentrate on problem-solving not the language itself - harness the power of the computer
- Lots of built-in functionality and support libraries ("batteries included")
- General purpose, multiple paradigm language and syntax support straightforward transition to C++, Java
- Popular, well-supported, good documentation and development environments.

For Python (or any computer language):
- What are the primitive elements?
- How can we combine the primitive elements?
- How can we extend the language?

Python primitive elements are called

Objects – value, type, id

For example, the number 4 is a Python object.

Python types

Values
E.g., Numbers (0, -7, 5.0, 1000, 999999)

Operations
for example: +, -, *, / 
also: //, %, **, pow, round
Python primitive elements

- **Types** describe a set of values, along with a set of operations that can be performed on those values.

Objects can be combined in **expressions**
- e.g., ```>>> 44``` or ```>>> 44 + 55```  
Expressions are evaluated and return a **value**

Python Primitive Elements

Variables can be named

Variables

Why variables?

Readability/Clarity

Reuse

```python
>>> a = 42
>>> b = a
>>> a = a * 2
>>> a
```

```python
>>> a = 10
>>> b = 20
>>> a + b
```
Solving a problem using Python:

• you have $50 and are buying some movies that cost $15 each. Write an expression that shows how much money you have left after buying n movies. Evaluate the expression when n = 2 and n = 3.

```python
>>> x = 0
>>> x = x + 1
>>> x = x + 2
>>> x

start_cash = 
movie_cost =
 n =
end_cash =
```