Welcome to CIS 210

What is Computer Science?

- the study of algorithms (MR p. 2)
- the study of problems, the [computational] problem-solving process, and the solutions [algorithms, programs] to those problems. (MR p. 330)

An algorithm is a sequence of well-defined operations.

FOR EXAMPLE: Fizzbuzz

0. form into groups of 3-5 students
1. the first person says the number 1
2. go around the group, with each person saying the next number in turn
3. though if the number is divisible by 3, say “fizz”, and if the number is divisible by 5, say “buzz”, and if the number is divisible by 3 and 5, say “fizzbuzz”
4. if an error is made, start again
5. stop when you reach 100

Computational problem solving is an algorithms-based approach to problem solving that is inspired and constrained by the possibilities and limitations of computers and computing.
ALGORITHMS — structured solutions to problems –

- have been around for a long time
- use an existing one
- adapt (revise, refactor) an existing one
- develop a new one

can be carried out (implemented) by a person or a computer

Computational process: an algorithm that can be implemented on a computer.

Computational Problem Solving

**TASK/PROBLEM** ➔ **Computational Thinking** ➔ **ALGORITHM/COMPUTATIONAL PROCESS**

**ALGORITHM** ➔ **Design/Coding** ➔ **COMPUTER PROGRAM**

**COMPUTER PROGRAM** ➔ **Testing/Debugging** ➔ **HIGH QUALITY COMPUTER PROGRAM**

**HIGH QUALITY COMPUTER PROGRAM** ➔ **Execute** ➔

AUTOMATIC, FAST, RELIABLE, REUSABLE

COMPUTER-GENERATED SOLUTION

Enlisting a computer as a problem-solving partner requires addressing the limitations of computers.
Enlisting a computer as a problem-solving partner requires addressing the limitations of computers.

We need tools, skills, approaches for communicating with computers for computational problem solving.

What is Computer Science?

- Theory
  - automata theory
  - algorithms & data structures
  - complexity
  - programming languages

- Systems
  - computer organization
  - operating systems
  - networks/high performance computing/security

- Software Development/Engineering
  - programming best practices
  - programming large, complex systems

- Applied Computer Science
  - data analytics
  - computing + X (e.g., biology, linguistics, law, economics, etc.)

CIS 210 Computer Science I

Welcome to CIS 210

✓ What is Computer Science?
✓ Computational Problem Solving
  • What can you expect from CIS 210?
  • Prior programming experience/Quiz

What can you expect from CIS 210

CIS 210 Focus: Computational Problem Solving

✓ Weekly projects and exercises to support learning of computational problem solving and other computer science topics

Supported by

✓ Class – large group meetings, concepts, discussion, clickers
✓ Lab (KLA B26) – small group exercises, computers
✓ Class notes, project solutions, text readings and exercises – review, practice, explore
✓ Lab help hours – daily drop-in help
✓ Code review – 2+ per term
✓ Code demonstration – 1 per term
✓ Assessments – midterm and final exams
✓ Class Encore – weekly guided study groups
Welcome to CIS 210

What is Computer Science?
Computational Problem Solving
What can you expect from CIS 210?
Prior programming experience

Your questions
Quiz

Hello, Python and 210 style guidelines (Project 1)
CIS 210 / Welcome

210, 211, 212 Computer Science I, II, III
prereqs:
- Math 112 (231, 251 readiness)
- prior programming experience

familiar with programming fundamentals – variables, expressions, basic data types (e.g., numeric, boolean, string, collections), conditionals, loops, user-defined functions, parameter passing, ...

familiar with process of programming – time, bugs, technical glitches, multiple drafts ...

CIS 210 Computer Science I

Hofstadter’s Law: It always takes longer than you expect, even when you take into account Hofstadter’s Law.

Douglas Hofstadter, Gödel, Escher, Bach: An Eternal Golden Braid

Welcome to CIS 210

What is Computer Science?
Computational Problem Solving
What can you expect from CIS 210?

Prior programming experience

Your questions
Quiz
Hello, Python and 210 style guidelines (Project 1)

“Talk is cheap. Show me the code.”
A computer program implements an algorithm (computational process) on a computer.

A computer program is (therefore) a set of instructions written in a language the computer can understand.

What kind of language is that?

0s and 1s?

Natural language?
A program is a set of instructions written in a language the computer can understand.

What kind of language is that?
0s and 1s?
Natural language?

**Compromise: Programming languages**

Python (or any computer programming language) provides a notation (syntax, semantics) for writing out a computational process as a series of steps.

- Formal
- Precise
- Unambiguous
- Readable: “Code is more often read than written.” — Guido Van Rossum

The programming language we use provides a (relatively) high level method for communicating with the computer.

Theoretical underpinnings of computer science: any “Turing complete” language has the same functionality as any other language.
**Why Python?**

- Python is modern, high level language; widely used in many fields
- Accessible to entry level programmers and also for experts – like chess or tennis
- Interactive (interpreted), syntax-lite language – concentrate on problem-solving rather than the language itself; can easily test snippets of code
- Lots of built in functionality and support libraries (“batteries included”)
- General purpose, multiple paradigm language and syntax support straightforward transition to C, C++, Java
- Popular, well-supported, good documentation and development environments.

**Python/IDLE**

- Downloads along with Python
- Simple integrated development environment (IDE)
- Shell for exploring Python and testing bits of code
- Editor supports Python program development

**Python Quick Overview**

- Python language
- Python interpreter (program)

→ What sorts of Python language input does the Python interpreter recognize/“understand”?

**Python (and any programming language)**

-- **keywords**
-- **primitive elements**
-- **identifiers**

**Python keywords, primitive elements, identifiers**

Keywords such as `def`, `;`, `=`, `while`, `if`, `return` -

- structure code
- store/retrieve values
- indicate order statements are executed
- special operations

Keywords are fixed, i.e., part of the language. We can use but not create them.
CIS 210 / Welcome

>>> help()
Welcome to Python 3.6's help utility!
...
help> keywords
Here is a list of the Python keywords. Enter any keyword to get more help.
False  def  if  raise
None   del  import  return
True   elif  in  try
and    else  is  while
as     except lambda with
assert finally nonlocal yield
break  for  not
continue from  or
class  global  pass
help>

Python ✓ keywords, primitive elements, identifiers

Keywords define the syntax and structure of the language. They are fixed, i.e., we use keywords but cannot create new keywords.

Primitive elements are basic objects of the language, for example, integers, strings, and functions.

They can be combined to create new elements.

3 important questions you should ask about any programming language:

• what are the primitive elements?
• how can we combine elements?
• how can we create our own elements?

many Python primitive elements – objects – are available when Python starts

>>> 4
# objects have values
4
# literals evaluate to themselves

>>> 'hello'
'hhello'

>>> len
# len is a built-in function
<len>

many Python primitive elements – objects – are available when Python starts

>>> type('hello')
# objects have types
<class 'str'>

>>> type(4)
<class 'int'>

>>> type(len)
# len is a built-in function
<class 'builtin_function_or_method'>
Python Shell/interpreter is a REPL.

>>> 4 + 3
7

Expressions are executed (evaluated) and return a value (which is printed when code is executed in Shell).

>>> len('hello')
5

>>> max(1, 2, 3)
3

>>> max(1, 2, 3) (evaluated) and return a value

>>> type(len('hi')) (which is printed when code is executed in Shell)

Python elements can be named – assignment

>>> my_identifier = 4  # Python assignment

compare to

>>> my_identifier  # Python expression
4  # returns a value

Python assignment statements are not expressions and do not return a value. They associate an identifier (name) with a value (object); this information is stored in a Python namespace.

Python organizes and keeps track of identifiers in namespaces.

When Python starts up, two namespaces are available:

__builtins__ - Python built-in objects

__main__ - objects defined during Python session
Python Functions

Executing, or calling, a built-in function:

```python
>>> abs()
TypeError: abs() takes exactly one argument (0 given)
```

```python
>>> abs(7)
7
```

#function calls are expressions
#they evaluate to a value
#(another Python object)

How can we extend the language – create our own elements?

- user-defined functions

For example
```python
def twice():
    # header incl. parameters
    # docstring
    result = 3 * 2
    return result
```

Defining a function is like defining a variable – the function name/identifier refers to the operations specified in the function definition.

Recall: Python functions name an operation, e.g., built-in function abs
```python
>>> abs
>>> help(abs)
<built-in function abs>
```

```python
>>> abs()
# functions are callable
```

For example
```python
def twice():
    # header incl. parameters
    # docstring
    result = 3 * 2
    return result
```

```
7
9
```

```
9
```

"syntactic sugar"
Functions are “callable”, or “executable”, data types – when a function is called, the operations specified in the function definition are executed.

Calling/executing a user-defined function is the same as calling/executing a built-in function:

```python
>>> twice()
6
```

```
def twice():
    """docstring goes here
    """
    result = 3 * 2
    return result

Calling/executing a user-defined function is the same as calling/executing a built-in function:

>>> twice()
6
```

```
def twice(n):
    """(number) -> number
    Returns double the value of input number n.
    """
    result = n * 2
    return result

>>> twice(3)
6
>>> twice(10.5)
21.0
```

```
def twice(n):
    """(number) -> number
    Returns double the value of input number x.
    """
    result = n * 2
    return result

>>> help(twice)
```

```python
>>> twice(3)
6
>>> twice(10.5)
21.0
```

```
```

```python
>>> help(twice)
```
Python toolkit so far

- Numeric data types (int, float) and operations (e.g., +, %, **, pow, round, abs)
- Boolean data types
- Strings
- Expressions
- Variables (identifiers)
- Assignment statement
- Python repetition – while
- User-defined functions: def, parameter list, docstring, code, return
- IDLE interactive development environment
- Python introspection – help, dir functions

[Tools in braces are in project 11-quiz example code only]