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 [programming] the solutions [algorithms] to those problems. (MR p. 330)

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.

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
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.

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

0, 1

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

science, math, design, engineering

What is Computer Science?
Support for/computational problem solving

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

science, math, engineering, design

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.)
Welcome to CIS 210

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

CIS 210 / Welcome

What can you expect from CIS 210

CIS 210 Focus: Computational Problem Solving
for a depth and breadth intro to computer science
  • Expand/improve software development skills
  • Demonstrate robust mental models of data representation and code execution
  • Introduction to Computer Science topics

CIS 210 Computer Science I

Welcome to CIS 210

• text 1st reading/practice – 1-2
• class/lab – 4
• wkly projects – understand/design/implement/test&debug – 6+
  [lab help hours]
• review text/class & lab notes/project solutions – 1-2

CIS 210

CIS 210 LEARNING OUTCOMES

• understand, develop, implement algorithms for computational problem solving;
• use structured design and testing methods to develop and implement computational solutions to problems (programs);
• read, write, revise, document, test, and debug algorithms and code;
• demonstrate robust mental models of data representation and code execution;
• demonstrate good understanding of a high level programming language;
• introduce and/or implement a sampling of classic computer science problem domains and algorithms.

What can you expect from CIS 210?

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

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
CIS 210 Computer Science I

What can you expect from CIS 210?
✓ Weekly projects and exercises to support learning of computational problem solving and other computer science topics
Supported by
✓ Class - large group meetings, concepts, discussion
✓ Lab (KLA B-26) - small group exercises, computers
✓ Class notes, lab exercises, project solutions, text readings and exercises - review, practice, explore
✓ Lab help hours - daily drop-in help
✓ Code review - 2+ per term
✓ Assessments - midterm and final exams
✓ Class Encore – weekly guided study groups

CIS 210 / Welcome

210, 211, 212 Computer Science I, II, III
prereqs: - Math 112
- 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

Welcome to CIS 210

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

Hello, Python

CIS 210 / Welcome

def testfunc(timeLeft, time):
    # prior programming experience
    # defining a function/parameters
    # [defining a function/parameters]
    # prior programming experience
    # [documenting code]
    # What does this function do?
    while timeLeft > 0:
        # [loops, expressions, Boolean type, number]
        # input/output, variables
        print(timeLeft)
        timeLeft = timeLeft - 1
        # [assignment, built in operators]
        # conditional
        if time == 'am':
            # string data type
            print('Good morning, CIS 210!')
            return timeLeft
    # function execution may have side effects
    # [functions return values]
    return

What will be the result when the following code is executed?

```python
>>> testfunc(10, 'am')
```

CIS 210 Intro to Computer Science

Class Encore:

Weekly guided study groups for challenging courses where students gather together once a week outside of class to practice exam concepts and strategies. Peer tutors share the class and assignment timelines for the group meeting. Participants in FBO and Spanish speaking students available in the data.

Students are encouraged to come every week. Weekly: Up to support the entire group. Space is limited. Register: Student study session.

Welcome to CIS 210

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

Hello, Python
“Talk is cheap. Show me the code.”

A computer program implements an algorithm 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?

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?

0 0
1 1
2 10
3 11
4 100
5 101
6 110
7 111
8 1000
9 ...

A  J  S...
B  K  T
C  L  U...
D  M  V...
E  N  W...
F  O  X...
G  P  Y...
H  Q  Z...
I  R...
Ok Google ...
Hey Siri ...
Good morning, Alexa ...

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.

computer levels of abstraction

Python is a language AND
Python is a program (translator/interpreter)
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.

Different programming languages -> easier or harder to accomplish different tasks.

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
What does this function do?

```python
def testfunc(timeLeft, time):
    # [input data is provided when function is called (executed)]
    while timeLeft > 0:
        # [flow of control, Boolean operation]
        print(timeLeft)
        # [display result]
        timeLeft = timeLeft - 1  # [variable/memory store]
    if time == 'am':
        # [flow of control, Boolean operation]
        print('Good afternoon, CIS 210!')
    return timeLeft  # [return result of executing testfunc]
```

What will be the result when the following code is executed?

```python
>>> testfunc(10, 'am')
```

---

**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

```python
>>> 4 #objects have values
4
>>> len
<built-in function len>
>>> 'hello'
'hello'
>>> type(4) #objects have types
<class 'int'>
```

Others are available in the Python standard libraries.

---

Python elements can be combined - expressions

```python
>>> 4 + 3 Python Shell/interpreter
7
>>> len('hello') #a REPL.
5
>>> max(1, 2, 3) (evaluated) and
3 return a value
>>> (which is printed when
code is executed in Shell)
```

---

elements can be named - assignment

```python
>>> 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.

---

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

– user-defined functions

For example

def twice():
    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.
>>> dir()
['__annotations__', '__builtins__', '__doc__', '__loader__', '__name__', '__package__', '__spec__', 'twice']

>>> twice
<function twice at 0x100560e18>

Calling a user-defined function is the same as calling a Python built-in function

>>> twice()
6

>>> help(twice)
(number) -> number

> number

Returns double the value of input number x.

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

Good programming style: function docstrings (type contract; description including parameters, returned value, examples of function use)

Python is a programming language and Python is an interpreter (program)

Python Shell is a REPL (read-evaluate-print loop)

Python keywords, primitive elements, identifiers

Combining primitive elements: Expressions - expressions evaluate to a value

Naming values: Variables/assignment - assignment statements are not expressions and do not return a value; namespaces - builtins, __main__

Functions are an executable (callable) data type
CIS 210

Python toolkit so far

numeric data types (int, float) and operations (e.g., +, **, //, %, round, abs) expressions variables (identifiers)/assignment statement Python keywords, e.g., def, return user-defined functions: def, parameter list, docstring, function code, return built in "introspection" functions, e.g., help, dir