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

A computational process is an algorithm that can be implemented on a computer.

Computational Problem Solving

PROBLEM/TASK →

COMPUTATIONAL PROBLEM SOLVING →

COMPUTATIONAL PROCESS, e.g.,
COMPUTER PROGRAM, i.e.,
COMPUTER-GENERATED SOLUTION/DESIRED OUTCOME

→ COMPUTER-GENERATED SOLUTION
Enlisting a computer as a problem-solving partner requires addressing the limitations of computers.
CIS 210 / Welcome

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

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

CIS 210 Learning Outcomes Promises

• 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.

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 / Welcome

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.)

CIS 210 Computer Science I

What can you expect from CIS 210?

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

Supported by

✓ Class - lectures – large group meetings
✓ Class - labs – small group meetings w/ computers
✓ Lab help hours/office hours – beyond class and lab
✓ Assessments, e.g., solutions, code feedback, exams
✓ Class Encore – CTL CIS 210 weekly study groups
Welcome to CIS 210

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

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 Approach
✓ What you can expect from CIS 210
  Prior programming experience/Quiz

Questions

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 ...

def testfunc(timeLeft, time):
    # prior programming experience
    ""
    What does this function do?
    ""
    while timeLeft > 0:
        print(timeLeft)
        timeLeft = timeLeft - 1
    if time == 'am':
        print('Good morning, CIS 210!')
    return timeLeft

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

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

CIS 210 / Welcome

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

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

```python
>>> testfunc(10, 'am') # [calling/executing a function/argument passing]
```
CIS 210 / Welcome

REVIEW ALL OF THE MATERIAL AT CLASS WEBSITE!

Required text / on reserve in library, including online

Schedule page – Weekly topic and readings, Class notes, Lab exercises, Projects, Project Solutions, Practice Exams, Exams, Solutions

Syllabus – Class policies, academic good conduct, how final grade is calculated, much more

Other Information – Lab help hours schedule, contact information, helpful and interesting links

https://classes.cs.uoregon.edu/18F/cis210/index.html
Canvas/Syllabus link

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

Your questions
Quiz

CIS 210 Computer Science I

Welcome to CIS 210

CIS 210/Questions

Hello, Python

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

**What kind of language is that?**

- Os and 1s?
- Natural language?

**Compromise: Programming languages**

**Python is a language AND**

Python is a program (translator/interpreter)

**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.

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"?
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?
```
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 -= 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')
```

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.

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.

Welcome to Python 3.6's help utility!
```
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 -= 1    # [variable/memory store]
    
    if time == 'am':  # [flow of control, Boolean operation]
        print('Good afternoon, CIS 210!')
    return timeLeft   # [return result of executing testfunc]
```

>>> help()
Welcome to Python 3.6's help utility!
```
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 -= 1    # [variable/memory store]
    
    if time == 'am':  # [flow of control, Boolean operation]
        print('Good afternoon, CIS 210!')
    return timeLeft   # [return result of executing testfunc]
```

Welcome to Python 3.6's help utility!
```
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 -= 1    # [variable/memory store]
    
    if time == 'am':  # [flow of control, Boolean operation]
        print('Good afternoon, CIS 210!')
    return timeLeft   # [return result of executing testfunc]
```

>>> help()
Welcome to Python 3.6's help utility!
```
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 -= 1    # [variable/memory store]
    
    if time == 'am':  # [flow of control, Boolean operation]
        print('Good afternoon, CIS 210!')
    return timeLeft   # [return result of executing testfunc]
```

Welcome to Python 3.6's help utility!
```
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 -= 1    # [variable/memory store]
    
    if time == 'am':  # [flow of control, Boolean operation]
        print('Good afternoon, CIS 210!')
    return timeLeft   # [return result of executing testfunc]
```

Welcome to Python 3.6's help utility!
```
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 -= 1    # [variable/memory store]
    
    if time == 'am':  # [flow of control, Boolean operation]
        print('Good afternoon, CIS 210!')
    return timeLeft   # [return result of executing testfunc]
```

Welcome to Python 3.6's help utility!
```
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 -= 1    # [variable/memory store]
    
    if time == 'am':  # [flow of control, Boolean operation]
        print('Good afternoon, CIS 210!')
    return timeLeft   # [return result of executing testfunc]
```

Welcome to Python 3.6's help utility!
Python keywords, primitive elements, identifiers

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?

Python elements can be combined - expressions

```python
>>> 4 + 3
7
>>> len('hello')
5
>>> max(1, 2, 3)
3
>>> Python Shell is a REPL.
```

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
>>> dir() ['__annotations__', '__builtins__', '__doc__', '__loader__', '__name__', '__package__', '__spec__', 'my_identifier']
>>> __name__ '__main__'
```
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.
09/28/18

**CIS 210** / Welcome

```python
>>> 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

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

**CIS 210 Introduction Computer Science**

```python
def twice(n):
    '''
    docstring goes here
    '''
    result = n * 2
    return result

>>> twice(3)
3 is an argument
6
>>> twice(5)
5 is an argument
10
```

**STYLE GUIDELINES**

```python
def twice(n):
    '''
    (number) -> number
    Returns double the value of input number n.
    Examples of use – always include at least two
    '''
    result = n * 2
    return result

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

Traceback (most recent call last):
  File "<pyshell#41>", line 1, in <module>
    twice()
  TypeError: twice() missing 1 required positional argument: 'n'

```python
>>> twice
<function twice at 0x105321158>
```
>>> help(twice)

    (number) -> number

    Returns double the value of input number x.

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