Welcome to CIS 122!
Intro to Programming and Computational Problem Solving

- What is computational problem solving?
- Why learn to program?
- What can you expect from CIS 122?
- Questions
- An example program

Computational Problem Solving

Get the computer to do the work for us

✓ automatic
✓ fast
✓ reliable
✓ reusable

solutions to a wide variety of tasks

How can we do this?

“There’s an app for that.”
Welcome to CIS 122
Intro to Programming and Computational Problem Solving

Get the computer to do the work for us

How can we do this?

*CIS 122:*
Using programs → Creating programs

**Computational problem solving**
is an approach to problem solving that is *inspired and constrained* by the *possibilities and limitations* of computers and computing

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

0, 1

Tools (e.g., language) for communicating w/ a computer

Structured approach to computational problem-solving provides context and foundation for using the tools

“The single most important skill for a computer scientist is problem solving.”
A structured approach to 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, ...

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/solution.

A computational process starts with a sequence of well-defined operations that moves from an initial starting point (task/problem) to a desired final outcome/solution.

FOR EXAMPLE: Fizzbuzz
0. form into groups of 4 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”, or if the number is divisible by 5, say “buzz”, or 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

A sequence of well-defined operations is called an algorithm.
Welcome to CIS 122
Intro to Programming and Computational Problem Solving

**ALGORITHMS**
- a sequence of well-defined operations
- a structured solution to a problem

- 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

Welcome to CIS 122
Intro to Programming and Computational Problem Solving

**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.

**TASK/PROBLEM →**
**ALGORITHM/COMPUTATIONAL PROCESS →**
**DESIRED OUTCOME/SOLUTION**

Welcome to CIS 122
Intro to Programming and Computational Problem Solving

**TASK/PROBLEM**
**ALGORITHM/COMPUTATIONAL PROCESS**
**DESIRED OUTCOME/SOLUTION (COMPUTER PROGRAM)**

A computational process starts with a sequence of well-defined operations (algorithm) that moves from an initial starting point to a desired final outcome implemented on a computer (computer program).

Welcome to CIS 122
Intro to Programming and Computational Problem Solving

**Computational Problem Solving**

**TASK/PROBLEM**
**Computational Thinking**
**EXAMPLES/DIAGRAMS/WORDS/...**
**UNDERSTAND/DESIGN AN**
**ALGORITHM/COMPUTATIONAL PROCESS**
**SOLUTION/COMPUTER PROGRAM**
Welcome to CIS 122
Intro to Programming and Computational Problem Solving

Computational Problem Solving

TASK/PROBLEM  Computational Thinking  EXAMPLES/DIAGRAMS/WORDS/...
UNDERSTAND/DESIGN AN ALGORITHM/COMPUTATIONAL PROCESS
ALGORITHM/C.P.  Design/Coding

**Design and coding (programming)** is the process of moving from an algorithm to an executable program.

Testing and debugging (more programming) (and other checking and cleaning up) is the process of moving from a program to a reliable, reusable program.

Recall:
Enlisting a computer as a problem-solving partner requires addressing the limitations of computers
0, 1

Tools (e.g., language) for communicating w/ a computer
✓ Structured approach to computational problem-solving
  → provides context and foundation for using the tools
  → also answers common intro to programming question:
  How do I start?
Welcome to CIS 122
Intro to Programming and Computational Problem Solving

✓ What is computational problem solving?
  • Why learn to program?
  • What can you expect from CIS 122?
  • Questions
  • An example program

(1) Gain a tremendously powerful problem-solving tool for your own subject discipline or personal use.

“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.”

(2) Programming as a liberal art:

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

(4) CIS 122 counts toward UO B.S. math/computing requirement, UO science group requirement, and as a CIS 210 programming “strongly encouraged” prerequisite.

Which CIS course is right for you?

110 Fluency with Information Technology / “Our Digital World”
111 Introduction to Web Programming
122 Introduction to Programming & Problem Solving
210, 211, 212 Computer Science I, II, III

✓ What is computational problem solving?
✓ Why learn to program?
  • What can you expect from CIS 122?
  • Questions
  • An example program
CIS 122 is a community of learners where

- Everyone is welcome
- Everyone is respected
- We value intellectual challenges and deliberate practice in pursuit of new knowledge and skills
- We support and encourage each other
- We celebrate our own and each other's accomplishments

Welcome to CIS 122
Intro to Programming and Computational Problem Solving

AFTER COMPLETING CIS 122
INTRO TO PROGRAMMING AND COMPUTATIONAL PROBLEM SOLVING,
YOU WILL BE ABLE TO ...

- use a computational problem-solving approach to generate computer solutions (programs) to problems
- write well-structured, well-documented programs using the Python programming language
- be prepared to continue to study programming or computer science on your own or in other courses

Welcome to CIS 122
Intro to Programming and Computational Problem Solving

What can you expect from CIS 122?

✓ Weekly projects and exercises to support learning of computational problem solving in a variety of areas

Supported by

✓ Class – large group – big picture concepts, exercises, Q/A
✓ Lab – small group – exercises, computers/whiteboards, Q/A
✓ Class notes (posted), project solutions (posted), text readings – review, practice, explore
✓ Help hours – daily drop-in help, tutorials, code reviews
✓ Comprehensive assessments – 2 midterm + 1 final "demo"
✓ Individual feedback on weekly projects

Welcome to CIS 122
Intro to Programming and Computational Problem Solving

YOU WILL FIND LOTS OF INFORMATION AT THE CIS 122 CLASS WEBSITE!

How to Get Help
- Lab help hours schedule

Class Schedule
- Weekly topics and readings
- Class notes
- Lab exercises/video
- Projects and solutions
- Practice exams and solutions
- Exams and solutions

Syllabus
- Learning outcomes
- Class policies
- How final grades are calculated
- and much more

Other Information including
- How to download and install Python
- Required text available at the UO Duckstore and online
- Links to other helpful and interesting resources

https://classes.cs.uoregon.edu/22W/cis122/
Canvas – use Syllabus link

Welcome to CIS 122
Intro to Programming and Computational Problem Solving

✓ What is computational problem solving?
✓ Why learn to program?
✓ What can you expect from CIS 122?
→ Questions
- An example program
An example program

```python
print('hello, world')
```

```python
uphill_pace = 30  # minutes per mile
downhill_pace = 20  # minutes per mile
uphill_distance = 1
downhill_distance = 2

uphill_time = uphill_pace * uphill_distance
downhill_time = downhill_pace * downhill_distance

total_time = uphill_time + downhill_time

average_pace = total_time / total_distance

average_pace = round(average_pace)

print('Average pace was', average_pace, 'minutes per mile.')
```

```python
def fb(n):
    '''(n: int) -> None
    An example program.
    >>> fb(15)
    '''
    for i in range(1, n+1):
        m3 = (i % 3) == 0
        m5 = (i % 5) == 0
        if m3 and m5:
            print('fizzbuzz')
        elif m3:
            print('fizz')
        elif m5:
            print('buzz')
        else:
            print(i)

    print('Game over!')
    return

>>> fb(15)
1
2
fizz
4
fizz
fizzbuzz
7
fizz
fizz
11
13
14
fizzbuzz
```

```python
# turtle
pencolor('purple')  # mark start of route on EMU lawn
speed(3)  # set the scene
reset()
rt(90)
fd(15)

bgpic('uo_campus_map.png')  # mark the spot
```

```python
print('hello, world')
```

```python
uphill_pace = 30  # minutes per mile
downhill_pace = 20  # minutes per mile

uphill_distance = 1
downhill_distance = 2

uphill_time = uphill_pace * uphill_distance
downhill_time = downhill_pace * downhill_distance

total_time = uphill_time + downhill_time

average_pace = total_time / total_distance

average_pace = round(average_pace)

print('Average pace was', average_pace, 'minutes per mile.')
```
What is computational problem solving?

Why learn to program?

What can you expect from CIS 122?

Questions

An example program

Hello, Python

- intro to programming languages/Python
- intro to IDLE development environment (lab)
- Python keywords, elements (objects), and identifiers
- naming objects – assignment
- assignment – reference diagrams
- combining objects to create new objects (expressions)

RECALL

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

0, 1

Tools (e.g., language) for communicating w/ a computer

Structured approach to computational problem-solving

provides context and foundation for using the tools

answers common intro to programming question: How do I start?

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

WHAT (KIND OF) "LANGUAGE" IS THAT?

Natural language?

On/off switches? 0s and 1s?

Programming language

Ok Google ...

Hey Siri ...

Good morning Alexa ...
Welcome to CIS 122
Intro to Programming and Computational Problem Solving

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 will use in 122, Python, provides a (relatively) high level method for communicating with the computer.
Welcome to CIS 122
Intro to Programming and Computational Problem Solving

computer levels of abstraction hide the details

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.

Welcome to CIS 122
Intro to Programming and Computational Problem Solving

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

Python/IDLE (Lab)
• In CIS 122 we will use IDLE to write and execute Python programs
• IDLE downloads along with Python
• Basic integrated development environment (IDE) supports writing Python code
• Shell for exploring Python, testing bits of code, executing a program – interacting with the Python interpreter directly
• Editor supports Python program development; code can be saved in a .py file

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, JavaScript, for example
• Popular, well-supported, good documentation and development environments.

Python/IDLE (Lab)
• A Python program is a sequence of Python definitions and commands that can be executed by the Python interpreter
• Every Python statement tells Python to do something.
• In the Shell – code runs immediately – the Shell is a Read-Evaluate-Print Loop (REPL).
• Need to Run Module from the IDLE Run menu to execute code in the Editor (.py file).
Welcome to CIS 122
Intro to Programming and Computational Problem Solving

Hello, Python

✓ intro to programming languages/Python
✓ intro to IDE development environment (lab)
  • Python elements (objects), identifiers, and keywords
  • objects have values and types
  • combining objects to create new objects (expressions)
  • naming objects – assignment
  • assignment – reference diagrams

Python Overview

• Python language
• Python interpreter (program)

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

Python (and any programming language)
  – primitive elements
  – identifiers (names)
  – keywords

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

Python primitive elements, identifiers, keywords

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

>>> 10
10  
>>> 5.6
5.6
>>> 'hello'
'hello'

Python primitive elements, identifiers, keywords

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

not objects:

>>> hello
Traceback (most recent call last):
  File "<pyshell#28>", line 1, in <module>
    hello
NameError: name 'hello' is not defined

>>> $ SyntaxError: invalid syntax
Welcome to CIS 122
Intro to Programming and Computational Problem Solving

Python primitive elements, identifiers, keywords

>>> 10   >>> 5.6   >>> 'hello'
10      5.6      'hello'

Objects have values and types

>>> type(10)   >>> type(5.6)
<class 'int'>  <class 'float'>

>>> type('hello')
<class 'str'>

73

Welcome to CIS 122
Intro to Programming and Computational Problem Solving

Python primitive elements, identifiers, keywords

For example,

>>> 44 + 55
99

But not

>>> len(44)

Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    len(44)
TypeError: object of type 'int' has no len()

75

Welcome to CIS 122
Intro to Programming and Computational Problem Solving

Python primitive elements, identifiers, keywords

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

They can be combined to create new objects.

>>> 10 + 1
11

76

Welcome to CIS 122
Intro to Programming and Computational Problem Solving

Python objects can be combined to create new objects – expressions

>>> 4
4

>>> 10 + 1
11

>>> round(99.9)
100

The returned value is automatically printed only when the code is executed in Shell.

(otherwise need to use Python print function)
(as you have seen/will see in lab this week)

78
Identifiers are the names given to Python objects.

Some identifiers come with the language (they are “built in”); new identifiers can be created.

Identifiers must be unique.

How can we create an identifier?
How do we know it is unique?

```python
my_sum = 4 + 3.5  # assignment statement
```

assignment statement creates an identifier (name)

```
my_sum  # for a Python object
```

Assignment statements are not expressions. They do not return a value.
Welcome to CIS 122
Intro to Programming and Computational Problem Solving

How can we create an identifier?
How do we know it is unique?

Assignment statements are not expressions.
They do not return a value.

Assignment statements associate a name with a Python object.

my_sum = 4 + 3.5
assignment statement
for a new Python object

my_sum creates an identifier (name)
7.5

This information is stored in a Python namespace.

namespace
my_sum 7.5
your_sum 8.5

don't store

my_sum
is an identifier/name/variable

Why variables?
Readability/Clarity
Reuse

Python (the language/interpreter) keeps track for us!

A reference diagram helps us keep track, too.

namespace          object space
my_sum     7.5
your_sum   X   8.5

round = 4  # be careful!
>>> round 4

>>> round(5.3)
Traceback (most recent call last):
  File "<pyshell#25>", line 1, in <module>
    round(5.3)
TypeError: 'int' object is not callable

Welcome to CIS 122
Intro to Programming and Computational Problem Solving

How can we create a new identifier?
How do we know it is unique?

>>> round = 4  # be careful!
>>> round
4

>>> round(5.3)
Traceback (most recent call last):
  File "<pyshell#25>", line 1, in <module>
    round(5.3)
TypeError: 'int' object is not callable

Welcome to CIS 122
Intro to Programming and Computational Problem Solving

How can we create a new identifier?
How do we know it is unique?

```python
>>> round = 4
>>> round
4
>>> round(5.3)
Traceback (most recent call last):
  File "<pyshell#25>" , line 1 , in <module>  
    round(5.3)
TypeError: 'int' object is not callable

>>> round + 1
??
```

How can we create an identifier?
How do we know it is unique?

```python
>>> cis122 = 10
>>> CIS122 = cis122 + 1.5
>>> CIS122 = cis122 * 2
>>> cis122
??
>>> CIS122
(draw the reference diagram)
??
```

Python ✓ primitive elements, ✓ identifiers, keywords
• expressions are literal values, variables, or combinations of these (operators/built-in funcs)
• expressions are evaluated and return a value
• assignment statements associate a name with a Python object
• Python keeps track of identifiers in a namespace (reference diagram)

Built-in Functions The Python interpreter has a number of functions and types built into it that are always available. They are listed here in alphabetical order.
Welcome to CIS 122
Intro to Programming and Computational Problem Solving

→ Python ✓primitive elements, ✓identifiers, keywords

keywords such as

def
for
in
import
[etc.]

and also some symbols such as : and =

Welcome to CIS 122
Intro to Programming and Computational Problem Solving

→ Python ▶primitive elements, ▶identifiers, ▶keywords

keywords such as def, for, in, import, and also some symbols such as : and =

• structure code
• store/retrieve values
• indicate the order statements are executed
• designate special operations

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

Welcome to CIS 122
Intro to Programming and Computational Problem Solving

→ Python ▶keywords, primitive elements, identifiers

keywords such as def, for, in, import, and also some symbols such as : and =

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

Welcome to CIS 122
Intro to Programming and Computational Problem Solving

def fb(n):
    '''(n: int) -> None
    An example program.
    >>> fb(15)
    '''
    for i in range(1, n+1):
        m3 = (i % 3) == 0
        m5 = (i % 5) == 0
        if m3 and m5:
            print('fizzbuzz')
        elif m3:
            print('fizz')
    …

Welcome to CIS 122
Intro to Programming and Computational Problem Solving

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

Python (and any programming language)
✓ primitive elements (objects)
✓ identifiers (name objects)
✓ keywords (structure code)
Welcome to CIS 122
Intro to Programming and Computational Problem Solving

Hello, Python
✓ intro to programming languages/Python
✓ intro to IDLE development environment (lab)
✓ Python elements (objects), identifiers, and keywords
✓ combining objects to create new objects (expressions)
✓ naming objects – assignment
✓ assignment – reference diagrams

Solve a problem using Python:
You have $50 and are buying some new movies that cost $15 each. You will also purchase some old movies, which are 1/3 the cost of a new movie. Write a Python program that shows how much money you would have left after buying 2 new movies and 2 old movies.

end_cash = 50 – (15 * 2) – (15/3 * 2)

... new movies actually cost $30
... [later] what problem is being solved ??

What do you think will happen when each line of code is executed?

a) 10 + 4
b) 10 - 4
c) 10 - 4

d) 10 + 4.0

e) -7
f) -7
g) 10 + 4 / 2
h) (10 + 4) / 2
Welcome to CIS 122
Intro to Programming and Computational Problem Solving

What do you think will happen when each line of code is executed?

1. \( \frac{2}{3} \)
2. \( \frac{2}{3} \)
3. \( \frac{1}{3} \times 15 \)
4. \( \frac{1}{3} \times 15 \)
5. \( (10 + 4) \mod 2 \)
6. \( 10 + 4 \mod 2 \)
7. \( 123 \div 10 \)
8. \( 123 \mod 10 \)
9. \( 2 \times 6 \)
10. \( 2 \times 2 \)
11. \( 2 \times 2 \)
12. \( 1 + 3 \times 5 \)
13. \( 2^6 \)

---

What do you think will happen when each line of code is executed?

1. \( \text{myvar} = 4 + 3 \)
2. \( \text{myvar} = 4 \)
3. \( \text{def} = 1.0 \)
4. \( \text{def} = 1 \)
5. \( \text{my_var} = 10 \mod 3 \)
6. \( \text{my_var} = 10 \mod 3 \)
7. \( \text{round} = 5 \)
8. \( \text{round} = 5 \)
9. \( \text{my_var} = 5 \times 4 \)
10. \( \text{my_var} = 5 \times 4 \)
11. \( \text{my_var} = 'hi' + 4 \)
12. \( \text{my_var} = 'hi' + 4 \)
13. \( \text{my_var2} = 99 + 1.0 \)
14. \( \text{my_var2} = 99 + 1.0 \)
15. \( \text{my_var2} = 1 + 2.5 \)
16. \( \text{my_var2} = 1 + 2.5 \)
17. \( \text{my_var2} = 101 \)
18. \( \text{my_var2} = 101 \)
19. \( \_ = 101 \)
20. \( \_ = 101 \)

---

What do you think will happen when each line of code is executed?

1. \( x = 0 \)
2. \( x = x + 1 \)
3. \( x = x + 2 \)
4. \( x = x + 2 \)

(hint: draw the reference diagram)

---

Python Primitive Elements

Variables

1. \( \text{example} = 88 \)
2. \( \text{example} = 88 \)
3. \( \text{example} = 99 \)
4. \( \text{example} = 99 \)

a) 88    b) 99    c) syntax error    d) none of these

---

Python Primitive Elements

Variables

1. \( \text{nextvar} = \text{example} \div 9 \)
2. \( \text{nextvar} = \text{example} \div 9 \)
3. \( \text{nextvar} = \text{example} // 9 \)
4. \( \text{nextvar} = \text{example} // 9 \)

a) 11    b) 11.0    c) syntax error    d) none of these

---

Python Primitive Elements

Variables

1. \( \text{nextvar} = \text{example} \div 9 \)
2. \( \text{nextvar} = \text{example} // 9 \)
3. \( \text{nextvar} = \text{example} // 9 \)
4. \( \text{nextvar} = \text{example} // 9 \)

a) 11    b) 11.0    c) syntax error    d) none of these
CIS 122 Learning Outcomes

- use a computational problem-solving approach to generate computer solutions (programs) to a variety of problems
- write well-structured, well-documented programs using the Python programming language
- be prepared to continue to study programming or computer science on your own or in other courses

CIS 122

Computational problem solving: from problem to algorithm/computational process to program to high quality program (automatic, fast, reliable, reusable)

Structured approach to computational problem solving aka programming =
- computational thinking = coding = testing/simulating
- Computers are binary machines (on, off), we need a way to communicate with them.
- Programming languages provide a relatively high-level way to do this.
- Computers levels of abstraction hide the details of the level below.

Python is a programming language at one level and an interpreter program one level down.

Python programming language, like all languages, has primitive elements (objects), identifiers, and keywords.
- Objects have values and types, e.g., numbers or string, that define a range of values and permissible operations
- Assignment statements associate an identifier (variable name) with an object. Identifiers are unique (unambiguous).
- Variable names are important for communicating and re-using values.
- Expressions (identifier, objects, combination of these) are evaluated and return a value.

Python toolkit so far
- numeric data types (int, float) and operations (e.g., +, -, *, /, //, %, pow)
- string data type and operations (e.g., str, len)
- variables (identifiers)
- assignment statement – associates an identifier (variable name) with an object
- reference diagrams – keep track of identifiers, which must be unambiguous
- expressions – combination of values, variables, and operators; a value or variable all by itself is an expression
- print() (built-in function)
- help() (built-in function)

IDLE interactive development environment to support Python coding
- IDLE Shell is a REPL (read-evaluate-print loop) – code is executed immediately, returned value, if any, is automatically printed in the Shell