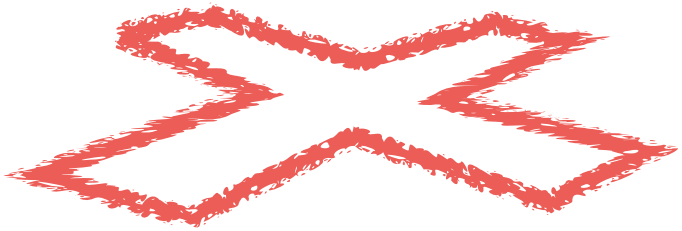
LIFE SKILL: PROGRAMMING & DEBUGGING

CSC111: Introduction to CS through Programming

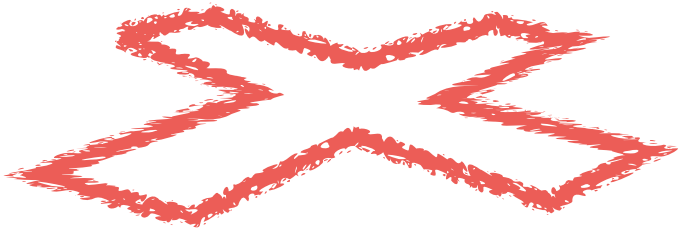
Slides from R. Jordan Crouser and Dominique Thiebaut
Smith College

The programming process



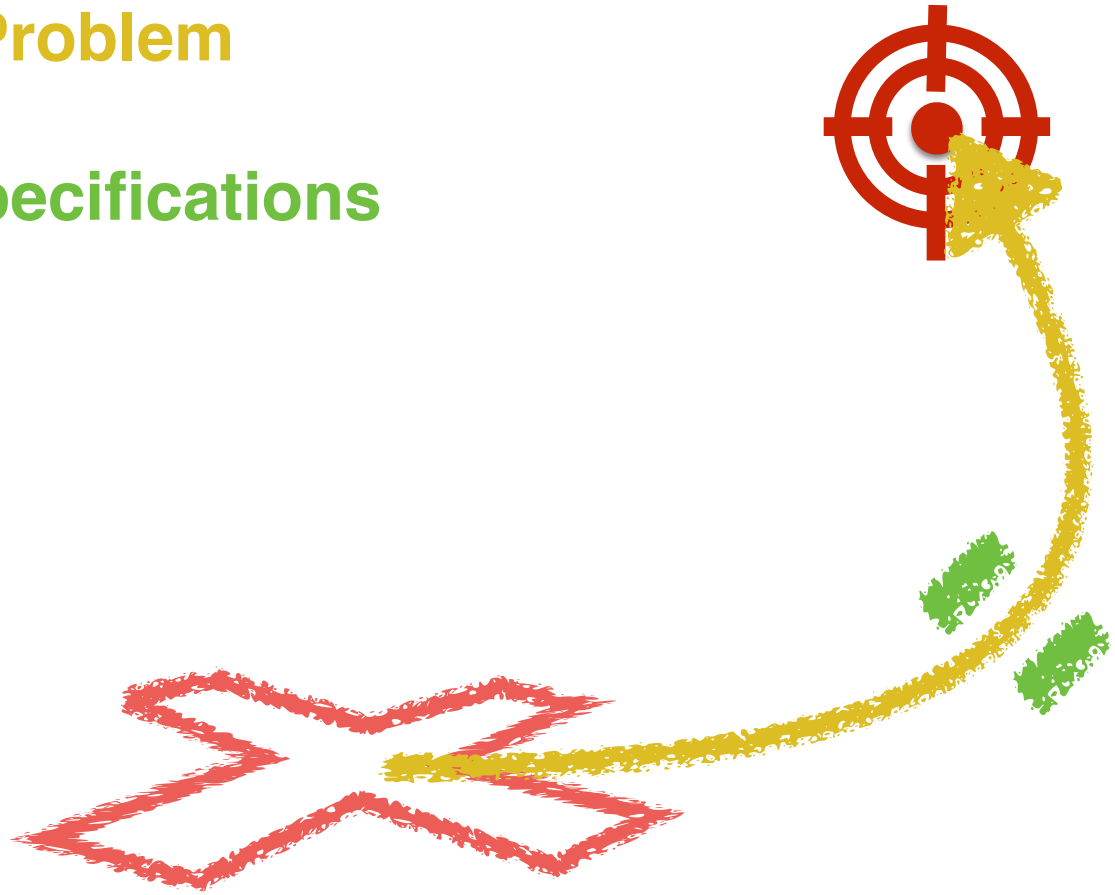
The programming process (idealized)

- Analyze the **Problem**



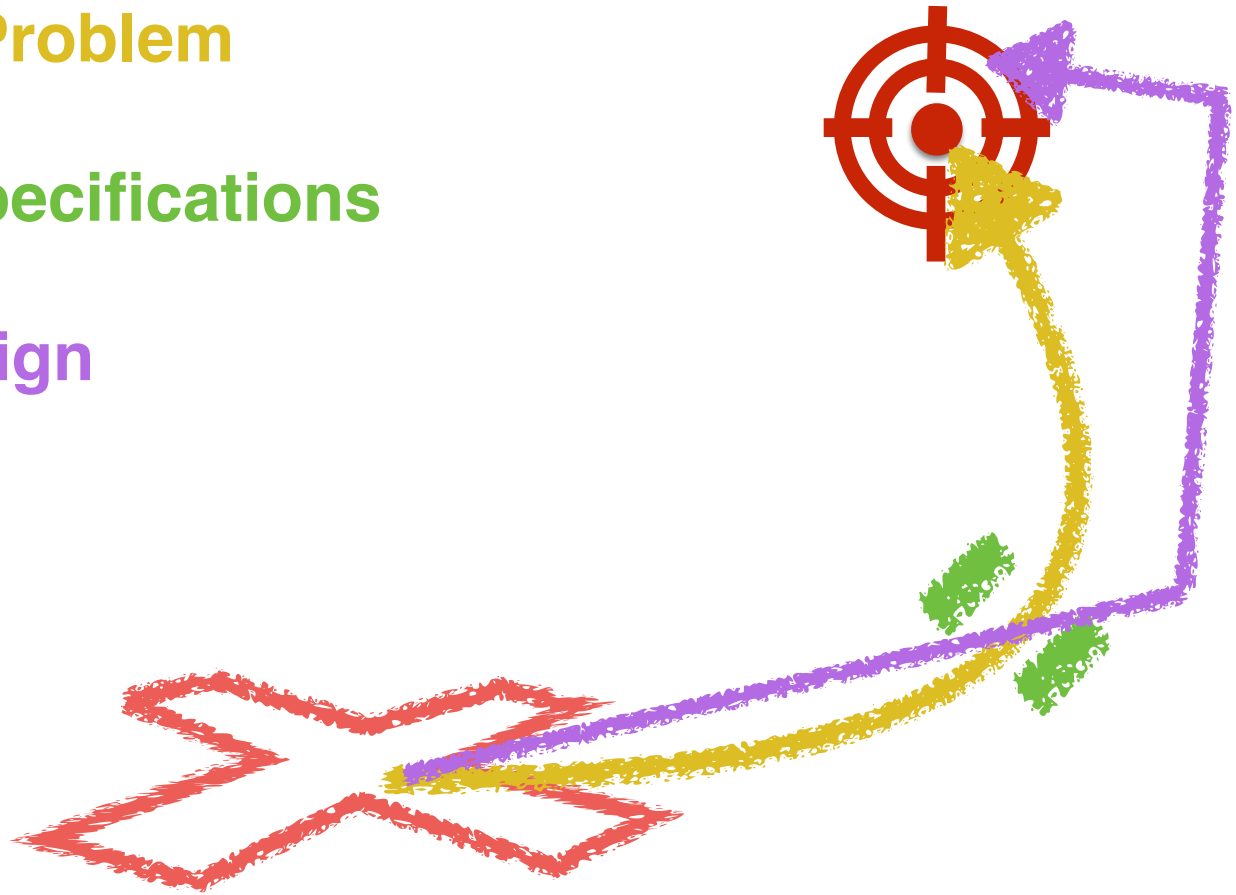
The programming process (idealized)

- Analyze the **Problem**
- Determine **Specifications**



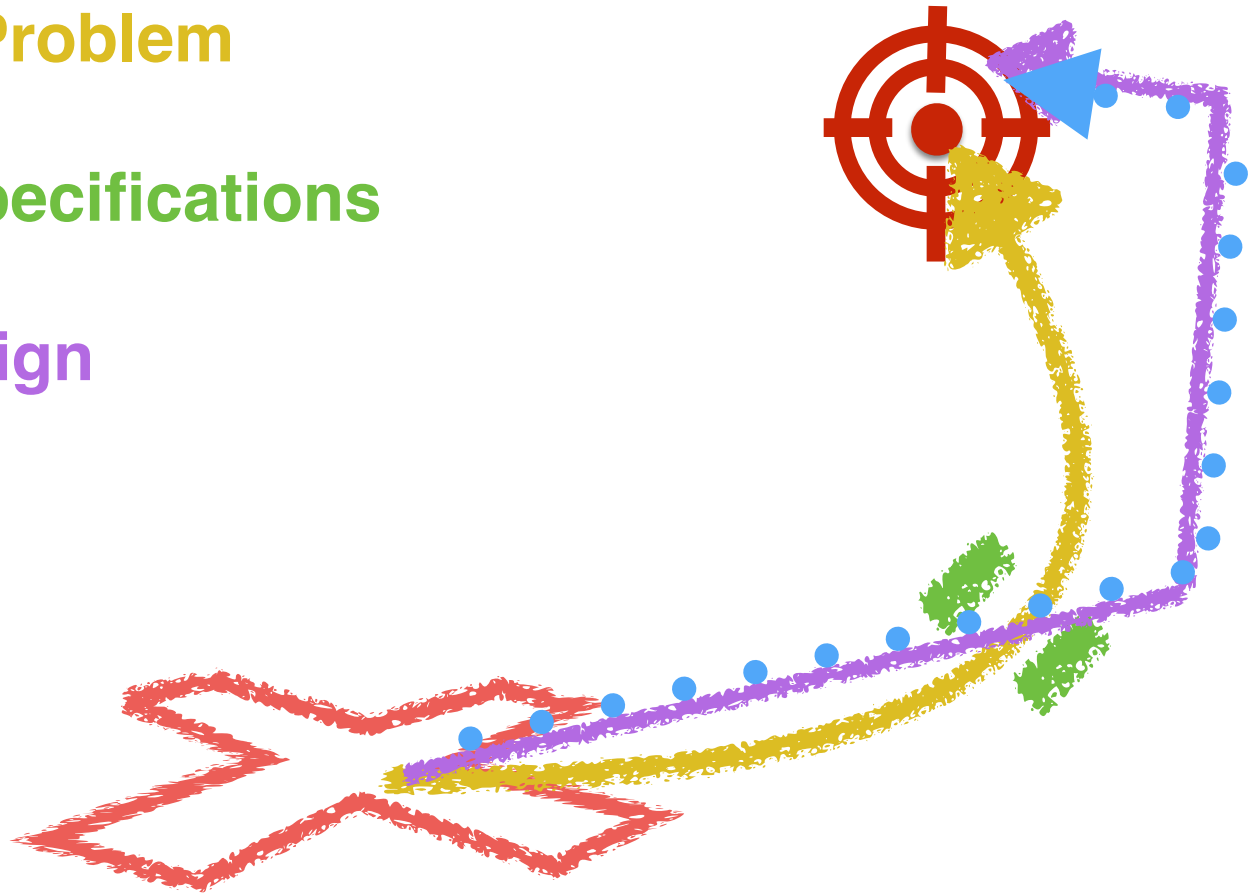
The programming process (idealized)

- Analyze the **Problem**
- Determine **Specifications**
- Create a **Design**



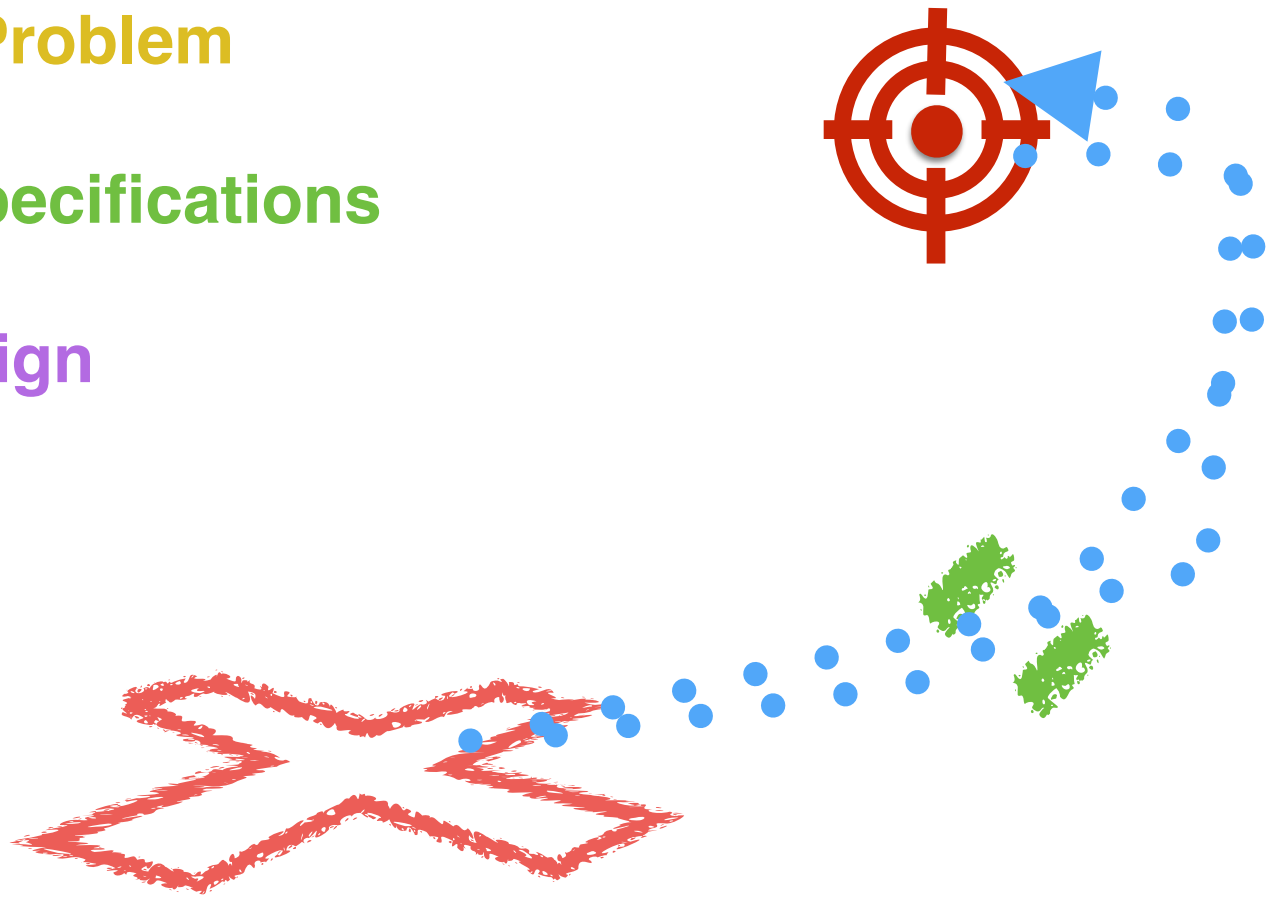
The programming process (idealized)

- Analyze the **Problem**
- Determine **Specifications**
- Create a **Design**
- **Implement**



The programming process (idealized)

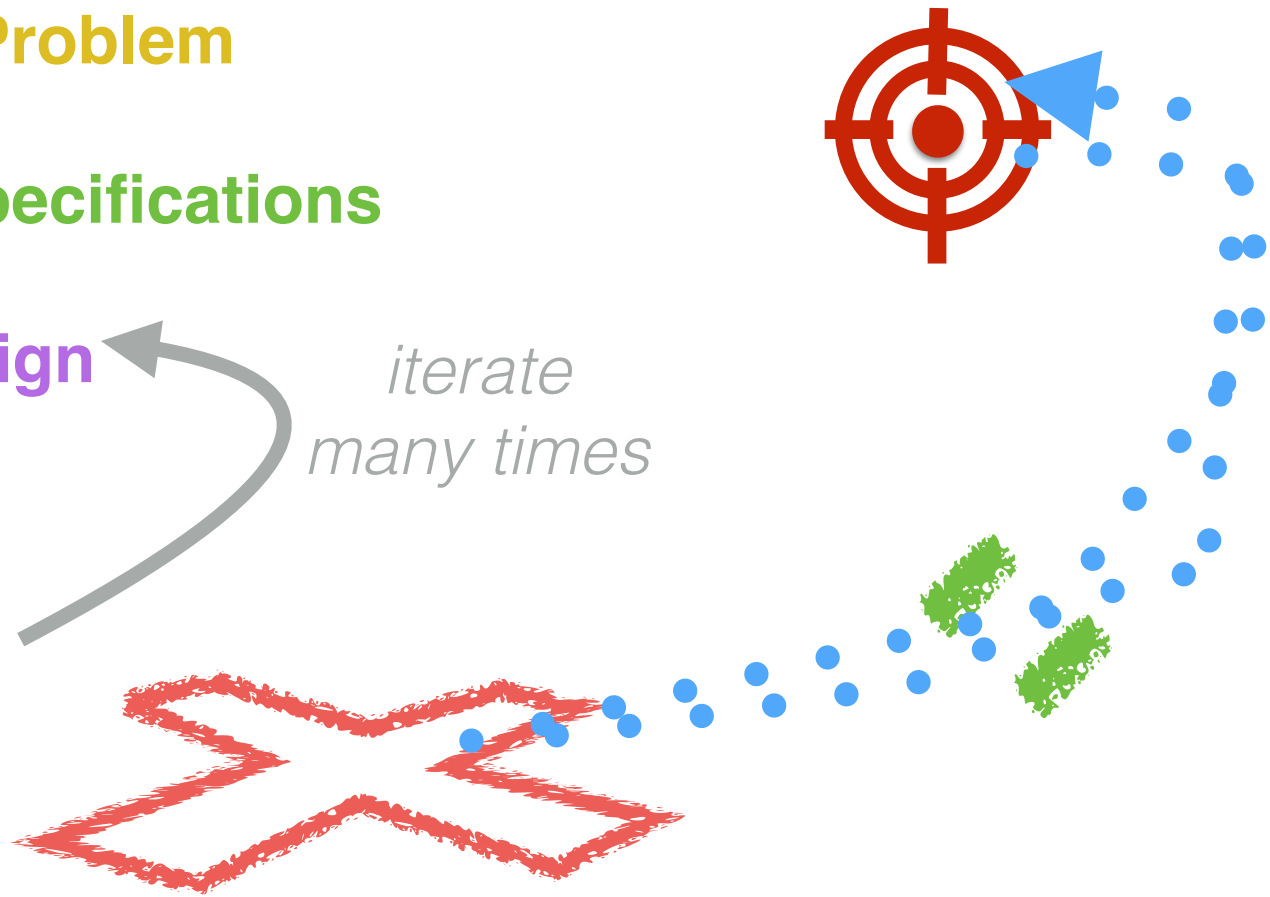
- Analyze the **Problem**
- Determine **Specifications**
- Create a **Design**
- **Implement**
- Test & Debug



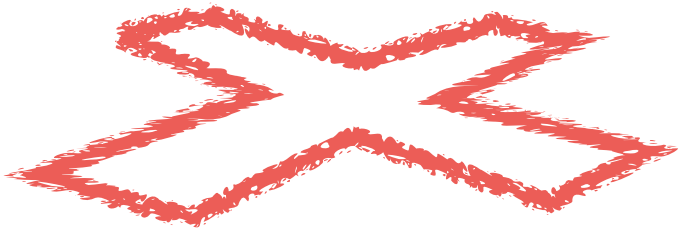
The programming process (more realistic)

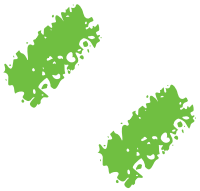
- Analyze the **Problem**
- Determine **Specifications**
- ~~Create a~~ Refine the **Design**
- **Implement**
- Test & Debug

iterate many times

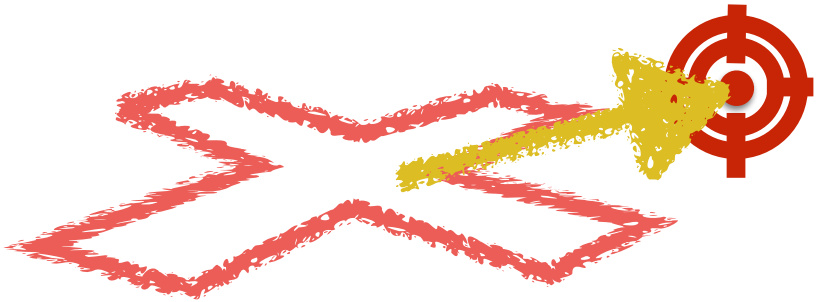


Getting started

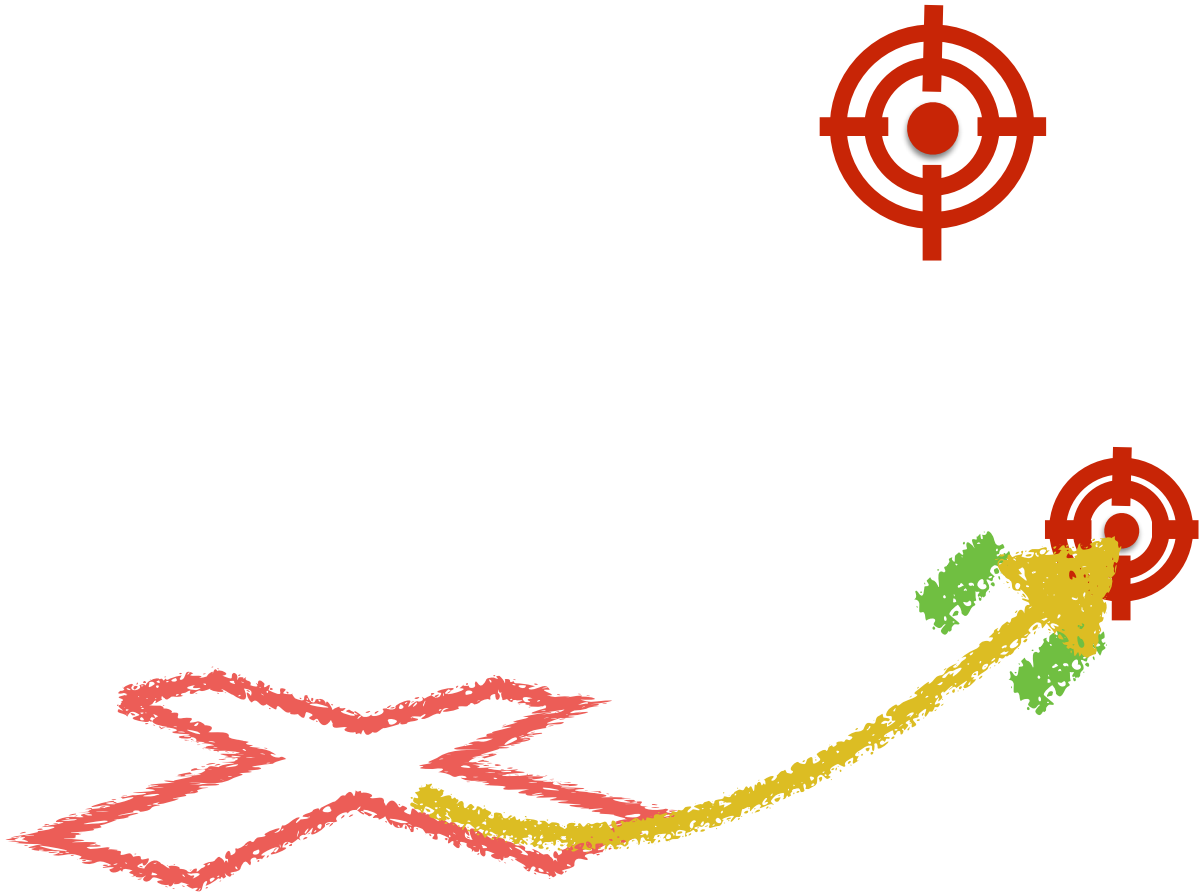




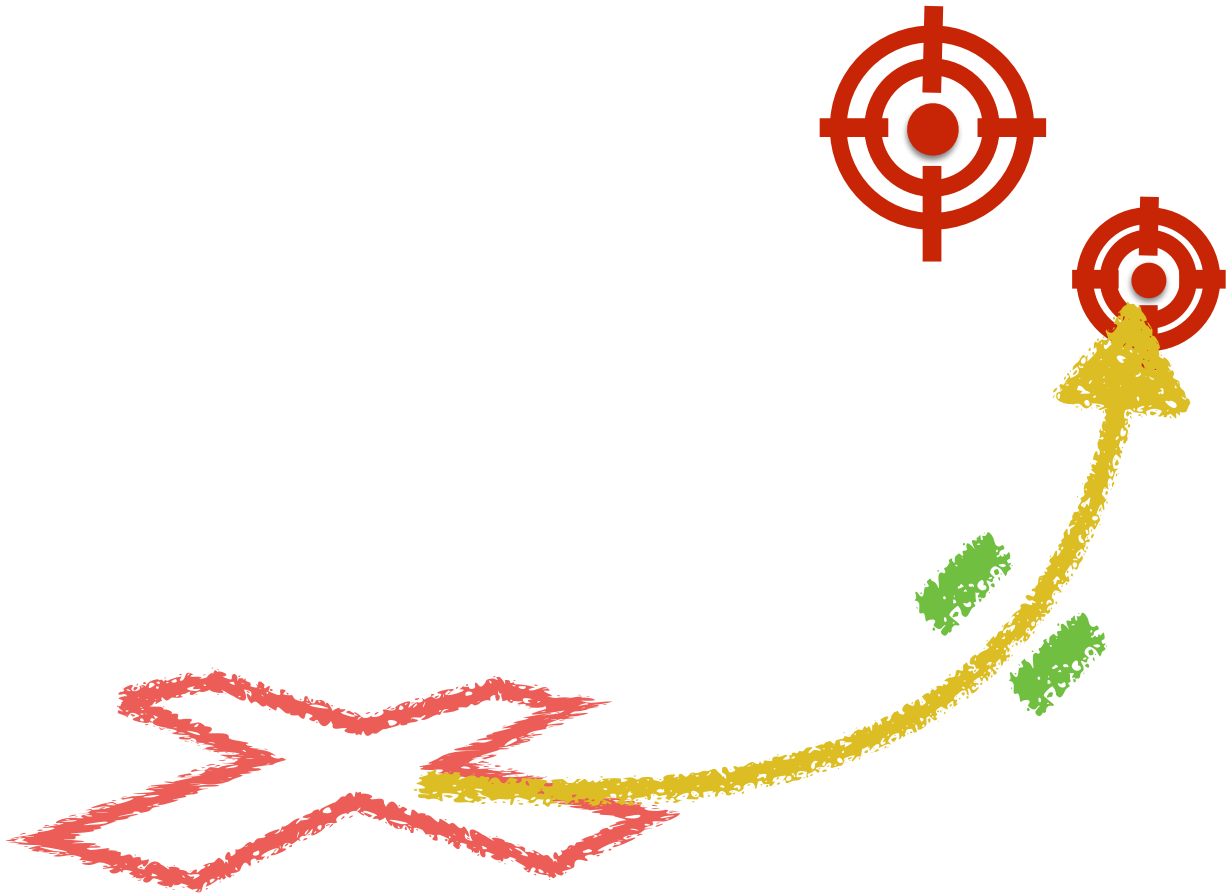
“S^4”: start small | slow | simple



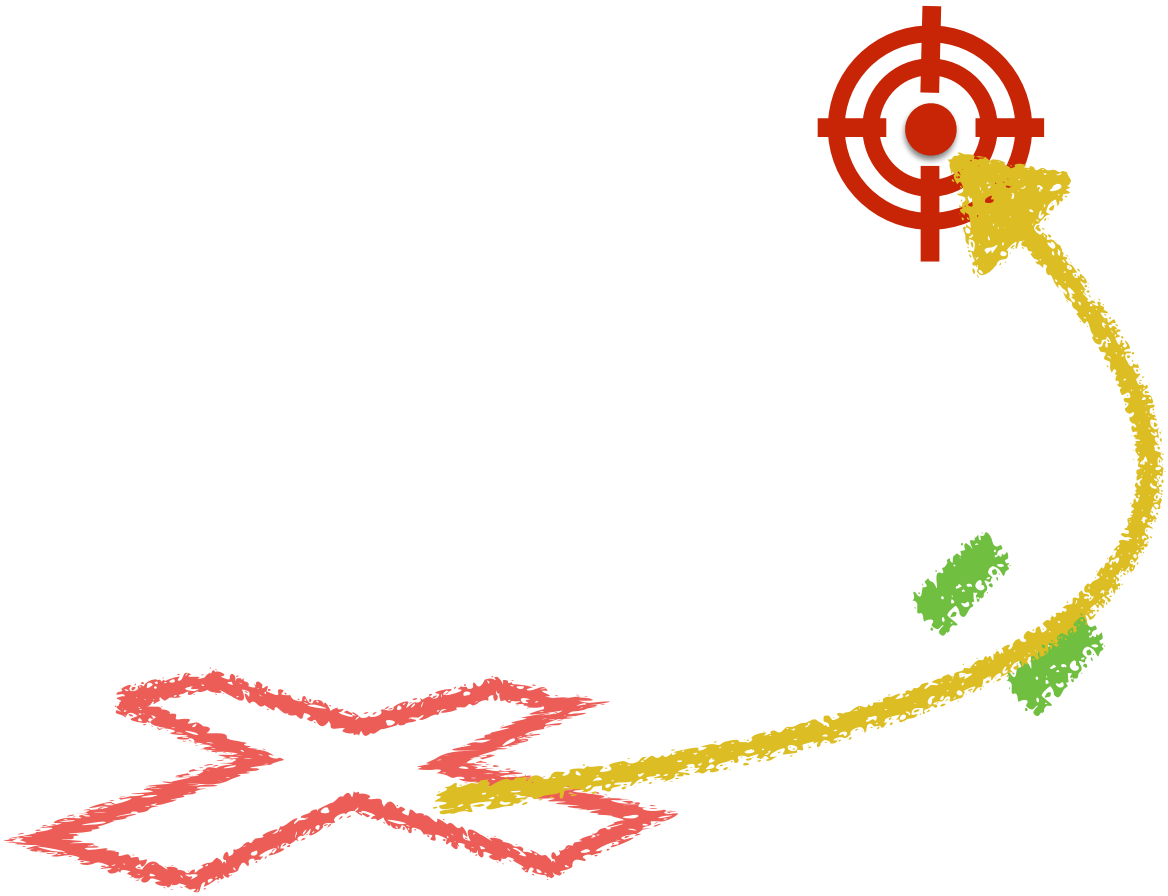
Next: address the constraints



Add additional features



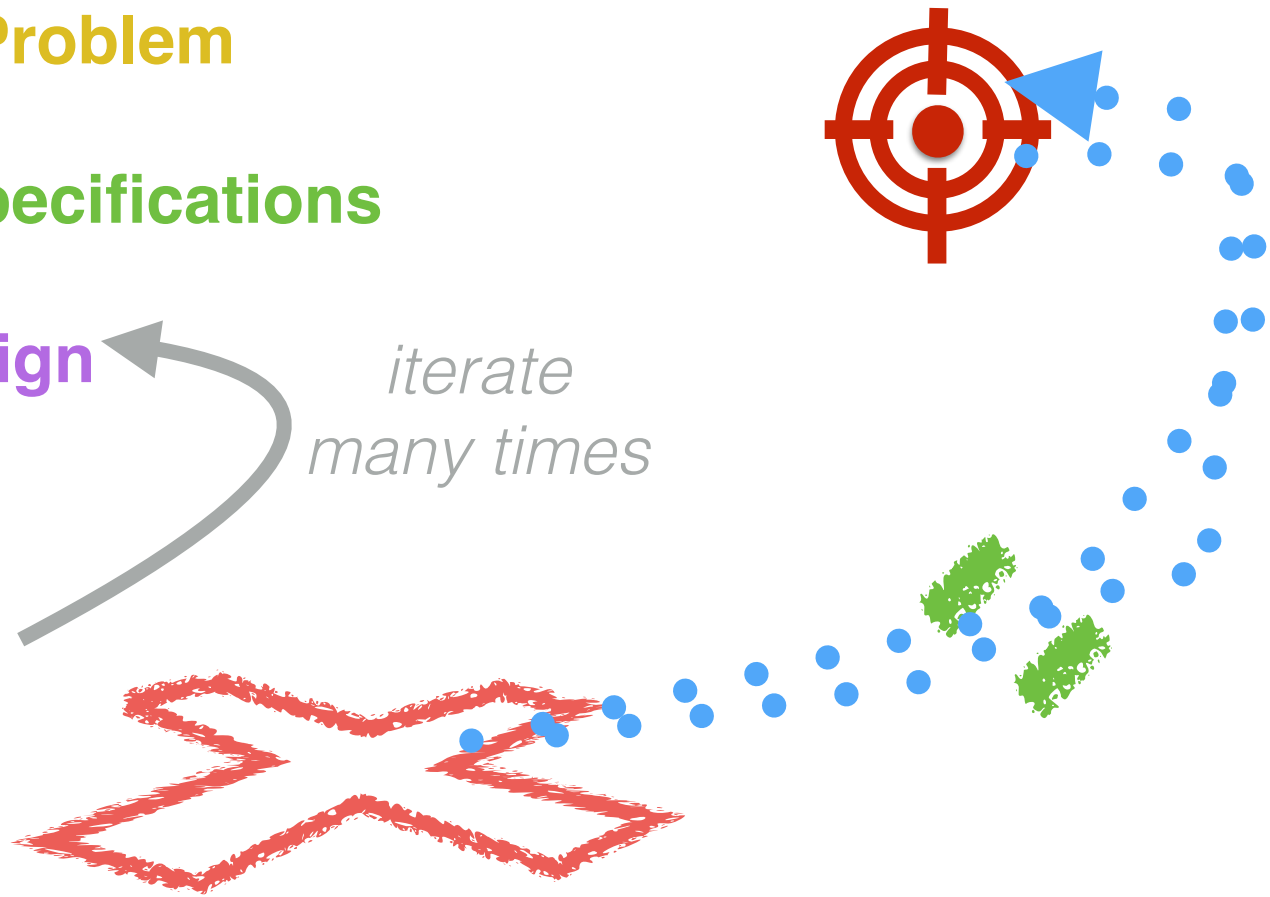
Finally: hit target



RECAP: the programming process

- Analyze the **Problem**
- Determine **Specifications**
- ~~Create a~~ Refine the **Design**
- **Implement**
- Test & Debug

iterate many times



Fun history: the term “debug”

9/9

0800 Arctan started
1000 " stopped - arctan ✓
13" vc (032) MP - MC
(033) PRO 2
correct
Relays 6-2 in 033 failed special speed test
In relay " 11.00 test.
Relays changed
1100 Started Cosine Tape (Sine check)
1525 Started Mult + Adder Test.
1545 Relay #70 Panel F
(moth) in relay.
First actual case of bug being found.
1630 Arctangent started.
1700 closed down.

RDML Grace M. Hopper

b.1906 – d.1992

Some problems are obvious

this is called
an **`Exception`**

Python 3.6.5 Shell

```
print(x)
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    print(x)
NameError: name 'x' is not defined
>>>
```

Ln: 11 Col: 4

Some problems are obvious

Python 3.6.5 Shell

```
print(x)
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    print(x)
NameError: name 'x' is not defined
>>>
```

Ln: 11 Col: 4

the kind of error gives you
a **clue** about what the problem is

Some problems are obvious

it also tells you **where** the problem is
(but be careful!)

Python 3.6.5 Shell

```
print(x)
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    print(x)
NameError: name 'x' is not defined
>>>
```

Ln: 11 Col: 4

Common **`Exceptions`**

- **`NameError`:** raised when Python can't find the thing you're referring to (a variable or a function)

Python 3.6.5 Shell

```
print(x)
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    print(x)
NameError: name 'x' is not defined
>>>
```

Ln: 11 Col: 4

Common **`Exceptions`**

- **`TypeError`**: raised when you try to perform an operation on an object that's not the right type (i.e. a `string` instead of a number)

Python 3.6.5 Shell

```
>>> 3 + "x"
Traceback (most recent call last):
  File "<pyshell#23>", line 1, in <module>
    3 + "x"
TypeError: unsupported operand type(s) for +: 'int'
and 'str'
```

Ln: 50 Col: 4

Common **`Exceptions`**

- **`IndexError`**: raised when you try to use an index that's out of bounds

Python 3.6.5 Shell

```
>>> x = ["a", "b", "c"]
>>> print(x[3])
Traceback (most recent call last):
  File "<pyshell#35>", line 1, in <module>
    print(x[3])
IndexError: list index out of range
```

Ln: 86 Col: 4

Common **`Exceptions`**

- **`SyntaxError`**: raised when you try to run a command that isn't a valid Python statement

```
*Python 3.6.5 Shell*
>>> print(hello my name is jordan)
SyntaxError: invalid syntax
>>>
Ln: 11  Col: 4
```

Common **`Exceptions`**

- **`SyntaxError`**: also raised if your indentation is messed up (this is a special kind of `SyntaxError` called an `IndentationError`)

Python 3.6.5 Shell

```
>>> if (x == 3):
        print(x)
        print("Done!")

SyntaxError: unexpected indent
>>>
```

Ln: 16 Col: 4

Common **`Exceptions`**

- **`ZeroDivisionError`**: raised when you try to divide by zero (or do modular arithmetic with zero)

Python 3.6.5 Shell

```
>>> x = 0
>>> y = 3 / x
Traceback (most recent call last):
  File "<pyshell#32>", line 1, in <module>
    y = 3 / x
ZeroDivisionError: division by zero
```

Ln: 75 Col: 4

Common **`Exceptions`**

- **`FileNotFoundError`**: raised when Python can't find the thing you're referring to (a file)

Python 3.6.5 Shell

```
>>> file = open("unicorn.txt", "r")
Traceback (most recent call last):
  File "<pyshell#33>", line 1, in <module>
    file = open("unicorn.txt", "r")
FileNotFoundError: [Errno 2] No such file or direct
ory: 'unicorn.txt'
```

Ln: 80 Col: 4

Common **`Exceptions`**

- **`UnicodeDecodeError`**: raised when you try to read a file that has weird characters in it (most common culprit: *apostrophe* vs. the *single quote*)

Python 3.6.5 Shell

```
>>> file = open("alice.txt", "r")
>>> file.read()
Traceback (most recent call last):
  File "<pyshell#30>", line 1, in <module>
    file.read()
  File "/Library/Frameworks/Python.framework/Versio
ns/3.6/lib/python3.6/encodings/ascii.py", line 26,
in decode
    return codecs.ascii_decode(input, self.errors)[
0]
UnicodeDecodeError: 'ascii' codec can't decode byte
0xe2 in position 219: ordinal not in range(128)
```

Less common **`Exceptions`**

Did your program throw an **`Exception`** not listed here?

Look it up at:

https://docs.python.org/3/library/exceptions.html

`Exceptions` = relatively easy to fix

Why would I say that?

What's the alternative?

Logical errors

- Mistakes in the **reasoning** behind the code (though the statements are valid and there are no `Exceptions`), e.g.

Untitled

```
x = ["A", "B", "C"]
choice = input("Enter A, B, or, C: ")
if choice == x:
    print("Okay!")
else:
    print("Invalid choice.")
```

Ln: 6 Col: 28

perfectly **valid**
(just not what we wanted)

Logical errors

- Mistakes in the **reasoning** behind the code (though the statements are valid and there are no `Exceptions`), e.g.

```
x = ["A", "B", "C"]
choice = input("Enter A, B, or, C: ")
if choice in x:
    print("Okay!")
else:
    print("Invalid choice.")
```

what we were **actually** going for

An analogy

Syntactic Error

Their is no reason to be concerned.

Logical Error

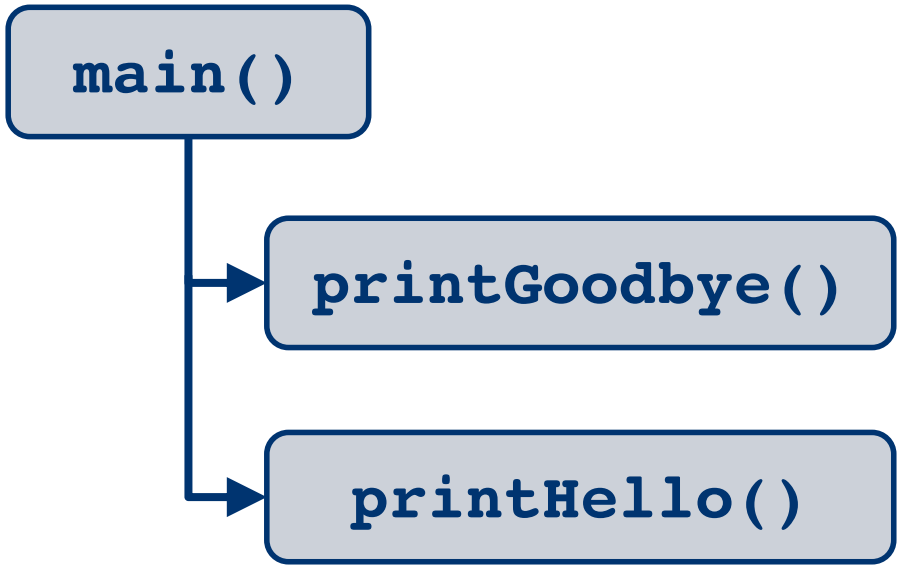
If an animal is green, it must be a frog.

Discussion

How do you find and fix **logical** errors?

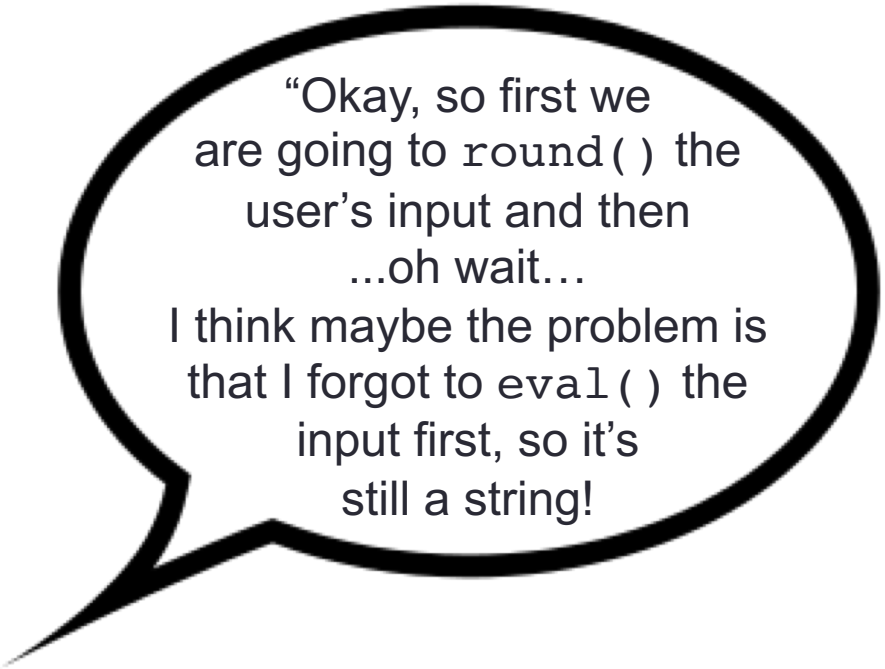
Step 1: map out the code

- It is impossible to debug code that you **don't understand** (and it's possible to not understand code even if **you wrote it**!)
- It's often helpful to map out how the code fits together:



Step 2: “rubber ducking”

- Still stuck? Try explaining it to someone else
  (or historically, to a rubber duckie)
- This is the debugging equivalent of **pair programming**



Step 3: add `print()` statements

- Not sure exactly where things are going wrong (esp. inside a loop)?
- Add `print()` statements to leave a “trail” on the console

Untitled

```
numbers = [1,2,3,4,5]
total = 0
for n in numbers:
    total + n

print(total)
```

Ln: 6 Col: 12

Takeaways

- This is a really quick crash course in **basic** debugging
- There are **lots** of other techniques for both dealing with and **preventing** bugs, but for now this will suffice
- The most important part is to understand:
  - what the code is **trying** to do
  - what the code is **actually** doing
- Tips:
  - change **one thing** at a time
  - **keep track** of what you change!

Demo

Your task

connectFour-broken.py - /Users/jcrouser/Google Drive/Teaching/Course Material/CSC111/CSC111/labs-old/connectFour-broken.py (3.6.5)

```
# ----------------------------------------------------
#       Names: <YOUR NAMES HERE>
#    Filename: connectFour-broken.py
#        Date: <TODAY'S DATE HERE>
#
# Description: This file contains a broken version
#              of Jordan's ConnectFour game.
#
#              There are 5 SYNTACTIC ERRORS (mistakes
#              that are not correct Python statements
#              and so cause the program to throw
#              Exceptions) as well as 5 LOGICAL ERRORS
#              (mistakes that are technically correct
#              Python statements, but which cause the
#              program not to behave the way we want).
#
#              Your job is to find (and correct!) each
#              of these mistakes using your new
#              DEBUGGING TECHNIQUES.
# ----------------------------------------------------
```

Ln: 15 Col: 54

Discussion

What did you find?