C++ as a First Language… Really?

Patrice Roy
Patrice.Roy@USherbrooke.ca
CeFTI, Université de Sherbrooke
Patrice.Roy@clg.qc.ca
Collège Lionel-Groulx

Who am I?

- Father of five (four girls, one boy), ages 24 to 6
- Feeds and cleans up after a varying number of animals
  - Look for « Paws of Britannia » with your favorite search engine
- Used to write military flight simulator code, among other things
  - CAE Electronics Ltd
- Full-time teacher since1998
  - Collège Lionel-Groulx, Université de Sherbrooke
  - Works a lot with game programmers
- Incidentally, WG21 and WG23 member (although I've been really busy recently)
  - Involved in SG12 and SG14, among other study groups
  - Occasional WG21 secretary
- And so on…

Who am I?

- Father of five (four girls, one boy), ages 24 to 6
- Feeds and cleans up after a varying number of animals
  - Look for « Paws of Britannia » with your favorite search engine
- Used to write military flight simulator code, among other things
  - CAE Electronics Ltd
- **Full-time teacher since1998**
  - **Collège Lionel-Groulx**, Université de Sherbrooke
  - Works a lot with game programmers
- Incidentally, WG21 and WG23 member (although I've been really busy recently)
  - Involved in SG12 and SG14, among other study groups
  - Occasional WG21 secretary
- And so on…

Who am I?

- Father of five (four girls, one b
- Feeds and cleans up after a var
  - Look for « Paws of Britannia »
- Used to write military flight si
  - CAE Electronics Ltd
- **Full-time teacher since1998**
  - **Collège Lionel-Groulx**, Université de Sherbrooke
  - Works a lot with game programmers
- Incidentally, WG21 and WG23 member (although I've been really busy recently)
  - Involved in SG12 and SG14, among other study groups
  - Occasional WG21 secretary
- And so on…

Typical student is between 17 to 20, and straight out from five years of high school, little to no programming experience.

Note: in Québec: six years of elementary school, five years of high school, then two or three years of college before university

Beliefs

Beliefs

- There is a widely held belief that C++ is, in no particular order:
  - A difficult language to learn
  - A complicated / complex language
  - A language inappropriate for beginners
  - Messy, in part due to its allegedly messy C heritage
  - An expert-friendly language
  - etc.

Beliefs

- There is a widely held belief that C++ is, in no particular order:
  - A difficult language to learn
  - A complicated / complex language
  - **A language inappropriate for beginners**
  - Messy, in part due to it llegedly messy C heritage
  - An expert-friendly langu
  - etc.

This talk focuses on this specific belief

Note that efforts to simplify C++ have been ongoing for a while now, and are highly interesting and fruitful in themselves, but are not what this talk is about

Methodology

Methodology

- We are going to compare solutions to a set of beginner problems
  - We're talking about beginners *to programming*
  - Don't expect programming acrobatics in this talk
- These problems are taken from an actual curriculum for beginners
  - Students of age 17-20, with a majority of 17 years old
  - Typical experience : high school + two months
- Most of these problems have been used for years
  - I went through them myself as a student *using Pascal* in the late 1980's, so be assured there is no C++ bias in the selection

Methodology

- For each of these problems, we'll examine simple solutions in a few languages
    - Pseudocode (Python would be close to this)
    - C++
    - C#
    - Java
- We'll examine, at each step, what new concepts and information are presented to beginners
    - Our aim is to see what each language asks of beginners trying to solve simple problems

Methodology

- Some things that you might like or not, but are not important to our discussion:
  - Braces and naming
    - I have tried to respect established practice in C# and Java
    - I have used my preferred style in C++
    - In practice, we use whatever style our teaching department chooses
    - You'll see uppercase symbolic constants due to the presence of pseudocode (this makes them stand out)

Methodology

- Some things that you might like or not, but are not important to our discussion:
  - Declare first, use later
    - This is tied to the fact that we use pseudocode with beginners
    - In pseudocode, we do not declare variables
    - This avoids silly mistakes on both sides
    - … and is not a practice that I would recommend past the beginning stages (see my other CppCon talk later this week!)

Methodology

- Some things that you might like or not, but are not important to our discussion:
  - Declare first, use later
    - This is tied to the fact that we use pseudocode with beginners
    - In pseudocode, we do not declare variables
    - This avoids silly mis akes on bo
    - … and is not a pr ce that I stages (see my other C on talk th

```
// pseudocode
x ← 3
Print x
```

```
// C++
#include <iostream>
using namespace std;
int main() {
   int x;
   x = 3;
   cout << x;
}
```

Methodology

- Some things that you might like or not, but are not important to our discussion:
  - Code examples with `float` instead of `double`
    - Data types are one of the first things beginners are confronted with
    - The very fact that we distinguish integrals and floating point numbers comes to some of them as a surprise
    - So does the fact that `4/3` yields `1`, not `1.333`
      - Pocket calculators lie when using integral arithmetic!
    - The fact that compilers are picky with types and issue warnings is something we use as a teaching tool

Methodology

- Some things that you might like or not, but are not important to our discussion:
  - There will be essentially no error management
    - That's just not something one should start with
    - There's too much to learn before going there, and these are *beginner* problems
    - … but it's important to get there!

Hello World

Hello World

- The intent:
  - This is not for my students, it's for us in this room (or watching from afar)
    - With students, I use the next example to start with
    - They have usually done some maths in high school, and find math-related problems less surprising
  - We forget how peculiar programming is when beginning

Hello World

```
Print "Hello World"
```

Hello World

```
Print "Hello World"
```

Note:

- We are introducing a primitive, `Print`. For many, this is magic
- We are using "quotes" for a character string. This is natural to us, but not every beginner gets why this is important, and many confuse ‘single quotes’, "double quotes"… and ‘’two single quotes”

Hello World

```
#include <iostream>
using namespace std;
int main() {
   cout << "Hello World" << endl;
}
```

Hello World

```
#include <iostream>
using namespace std;
int main() {
    cout << "     lo World" << endl;
}
```

Two elements (the #include directive, and the using namespace std; line) do not make sense at first to a beginner who has no idea what programming is

Hello World

```
#include <iostream>
using namespace std;
int main() {
   cout << "Hello World" << endl;
}
```

There is a basic structure to main(). Indentation can be taught as a simple rule for now. So does that ‘int’ thing

Hello

```
#inclu
using namespace st
int main() {
    cout << "Hello World" << endl;
}
```

We are using words such as cout and endl, as well as the << symbol
We are using quotes for text
The ';' plays a grammatical role

Hello World

```
using System;
namespace Hello
{
  class Program
  {
    static void Main(string [] args)
    {
      Console.WriteLine("Hello World");
    }
  }
}
```

Hello World

```
using System;
namespace Hello
{
  class Program
  {
    static void Main(string [] args)
    {
      Console.WriteLine("Hello World");
    }
  }
}
```

Some elements (using System;, namespace Hello, class Program) do not make sense at first to a beginner who has no idea what programming is

Hello World

```
using System;
namespace Hello
{
  class Program
  {
    static void Main(string [] args)
    {
      Console.WriteLine("Hello World");
    }
  }
}
```

There is a basic structure to Main(). Indentation can be taught as a simple rule for now. So does that ‘static void’ thing

Hello World

We are using words such as Console.WriteLine as well as parenthesis

We are using quotes for text

The ‘;’ plays a grammatical role

```
using System;
namespace Hello
{
  class Program
  {
    static void Main(string [] args)
    {
      Console.WriteLine("Hello World");
    }
  }
}
```

Hello World

Some words (static, void, string) and the [] symbols stay magic for a while (note that we could remove the arguments to Main here, but this is what beginners are presented with)

```
using System;
namespace Hello
{
  class Program
  {
    static void Main(string [] args)
    {
      Console.WriteLine("Hello World");
    }
  }
}
```

Hello World

```
public class Hello {
   public static void main(String [] args) {
      System.out.println("Hello World");
   }
}
```

Hello World

```
public class Hello {
   public static void main(String [] args) {
      System.out.println("Hello World");
   }
}
```

Some elements (public class Hello) do not make sense at first to a beginner who has no idea what programming is

Hello World

```
public class Hello {
   public static void main(String [] args) {
      System.out.println("Hello World");
   }
}
```

There is a basic structure to main(). Indentation can be taught as a simple rule for now. So does that 'public static void' thing

Hello World

```
public class Hello {
    public static void main(String [] args) {
        System.out.println("Hello World");
    }
}
```

We are using words such as System.println as well as parenthesis

We are using quotes for text

The ';' plays a grammatical role

Hello World

```
public class Hello {
   public static void main(String [] args) {
      System.out.println("Hello World");
   }
}
```

Some words (public, static, void, String) and the [] symbols stay magic for a while

Hello World

- Regardless of language, programming can seem alien at first contact
  - It’s also fun and exciting, if you’re into that mindset!
  - One could claim that such or such syntax is less weird than some other
    - E.g. cout << "Hi"; or System.out.print("Hi");
  - Please remember that, for many at that stage, the function-like syntax with parentheses has never been used without “doing something” with the results (e.g. y=f(x))
    - It’s all fun and weird

Sequence, arithmetic and basic I/O

Sequence, arithmetic and basic I/O

- The intent:
  - At first, programming is almost like expressing a recipe for cooking, or driving directions that need to be precise
  - Some basic facts like "assignment flows from right to left" or "when going through a sequence of instructions, one has to do them one by one, in sequence" are not self-evident
    - In fact, assignment is new to most students at that age; they are used to y=f(x) meaning something quite different!
  - What follows is something I get to in the very first class of the semester

Sequence, arithmetic and basic I/O

- The way we do this:
  - Express the algorithm in pseudocode form
  - Test it on paper first
    - It's a matter of developing a mindset
    - I know some advocate using tools right away. I use tools in the very first class, but I push the "think before you code" aspect to avoid the (sadly, but understandably) very real "let's type until is seems to work" reflex beginners have
  - Pseudocode is generally simpler
    - Types and variable declarations are "not a thing" at that stage
    - Translating the general algorithm to an actual programming language involves inserting and understanding the details

Sequence, arithmetic and basic I/O

```
PI ← 3.14159
Read radius
volume ← 4/3 * PI * radius ^ 3
Print volume
```

Sequence, arithmetic and basic I/O

```
PI ← 3.14159
Read radius
volume ← 4/3 * PI * radius ^ 3
Print volume
```

There’s a *lot* of content here:

- Three primitives (Read, Print, ← for assignment)
- A symbolic constant (PI)
- Literal constants
- Input, output (some have trouble with this)
- Naming rules (we use some, to reduce confusion)
- Arithmetic operations, including priority and syntax
- Assignment is new to them, and flows from right to left
- It’s a sequence of operations (*don’t underestimate this!*)

Sequence, arithmetic and basic I/O

```
PI = 3.14159
radius = float(input("Radius? "))
volume = PI * 4/3 * radius * radius *
         radius
print(volume)
```

Sequence, arithmetic and basic I/O

```
PI = 3.14159
radius = float(input("Radius? "))
volume = PI * 4/3 * radius * radius *
         radius
print(volume)
```

I’m cheating to make this fit into the slide

Sequence, arithmetic and basic I/O

```
PI = 3.14159
radius = float(input("Radius? "))
volume = PI * 4/3 * radius * radius *
         radius
print(volume)
```

- Even Python is not as simple as pseudocode.
- Note that to do a simple computation, we need to do a type cast… with people who have no idea what a type is
- input("Radius? ") is not self-evident and requires explanations

Sequence, arithmetic and basic I/O

```
PI = 3.14159
radius = float(input("Radius? "))
volume = PI * 4/3 * radius * radius *
         radius
print(volume)
```

A word on static vs dynamic typing for beginners… I "learned" Python while preparing these slides, starting with only a basic graps of the language. It went well, but nothing worked the first time I tried it. The interactive console was cool, but error messages were very general, and typically not very helpful… Also, they popped up during execution in many cases. Your mileage may vary, but it's not self-evidently easier for a beginner

Sequence, arithmetic and basic I/O

```
#include <iostream>
using namespace std;
int main() {
   const float PI = 3.14159f;
   float radius,
         volume;
   cin >> radius;
   volume = 4.0f/3.0f * PI *
            radius * radius * radius;
   cout << volume << endl;
}
```

Sequence, arithmetic and basic I/O

```
#include <iostream>
using namespace std;
int main() {
   const float PI = 3.14159f;
   float radius,
         volume;
   cin >> radiu
   volume
                                                  ;

   cout <<
}
```

Constants and variables have to be declared (and defined) before use

Types appear (this is mistifying to many)

The ';' symbol

Sequence, arithmetic and basic I/O

```
#include <iostream>
using namespace std;
int main() {
    const float PI = 3.14159f;
    float radius
```

I typically write the literal without ‘f’ suffix first to get a warning and discuss the fact that there are more than one floating point type. We read the message the compiler gave us, and they tell me what I should do

Don’t underestimate the difficulty of that ‘reading’ part. It’s my biggest fight every year

```
                                                    us;

}
```

Sequence, arithmetic and basic I/O

Note the other floating point literals. I start by writing 4/3as in pseudocode (or Python), we make the thing compile, then we prepare a test plan.

We input 1 for radius, expecting to see 4.18 or so on screen, but we see 3.14159 instead. *This is typically a shock*, and leads some to awaken to what is going on

Only then do we change the literals, and show that 4.0f/3 would work too (which leads to discussing operator priority and types)

```
#i
us
ir

   cin >> radi   ;
   volume = 4.0f/3.0f * PI *
                radius * radius * radius;
   cout << volume << endl;
}
```

Sequence, arithmetic and basic I/O

```
#include <i
using names
int main()
   const fl
   float ra
         volume;
   cin >> radius;
   volume = 4.0f/3.0f * PI *
             radius * radius * radius;
   cout << volume << endl;
}
```

There is the translation from pseudocode to C++, which involves syntax:
Read x becomes cin >> x
Print x becomes cout << x (with endl to change line; the fact that we could want to change line or not is not self-evident)
x ← y becomes x = y

Sequence, arithmetic and basic I/O

```
using System;
namespace Sphère
{
   class Program
   {
      static void Main(string [] args)
      {
         const float PI = 3.14159f;
         float radius,
               volume;
         radius = float.Parse(Console.ReadLine());
         volume = 4.0f/3.0f * PI * radius * radius * radius;
         Console.WriteLine(volume);
      }
   }
}
```

Sequence, arithmetic and basic I/O

```
using System;
namespace Sphère
{
    class Program
    {
        static void Main(string [] args)
        {
            const float PI = 3.14159f;
            float radius,
                  volume;
            radius = float.Parse(Console.ReadLine());
            volume = 4.0f/3.0f * PI * radius * radius * radius;
            Console.WriteLine(volume);
        }
    }
}
```

Small detail, but: I'm French-speaking, and so are my students. They find being able to use accented characters in code to be a good thing

Sequence, arithmetic and basic I/O

```
using System;
namespace Sphère
{
    class Program
    {
        static void Main(string [] args)
        {
            const float PI = 3.14159f;
            float radius,
                  volume;
            radius = float.Pars          ole.ReadLine());
            volume = 4.0f/                                  ius;
            Console.Write
        }
    }
}
```

Constants and variables have to be declared (and defined) before use

Types appear (this is mistifying to many)

The ‘;’ symbol

Sequence, arithmetic and basic I/O

```
using System;
namespace Sphère
{
   class Program
   {
      static void Main(string [] args)
      {
         const float PI = 3.14159f;
         float radius,
```

I typically write the literal without ‘f’ suffix first to get en error and discuss the fact that there are more than one floating point type. We read the message the compiler gave us, and they tell me what I should do

```
   }
}
```

Sequence, arithmetic and basic I/O

```
using System;
namespace Sphère
{
    class Program
    {
        static voi
        {
            const f
            float r
                    volume;
            radius = float.Parse(Console.ReadLine());
            volume = 4.0f/3.0f * PI * radius * radius * radius;
            Console.WriteLine(volume);
        }
    }
}
```

Note the other floating point literals. I use the same technique as in C++ (4/3, then a test plan, then a discussion and a fix)

Sequen /O

```
using System;
namespace Sph
{
    class Prog
    {
        static
        {
            const float                  1159f;
            float radius,
                  volume;
            radius = float.Parse(Console.ReadLine());
            volume = 4.0f/3.0f * PI * radius * radius * radius;
            Console.WriteLine(volume);
        }
    }
}
```

There is the translation from pseudocode to C#, which involves syntax:

Read x becomes float.Parse(Console.ReadLine());

Print x becomes Console.WriteLine(x); (or Console.Write())

x ← y becomes x = y

Sequence, arithmetic and basic I/O

```
import java.io.*;
public class Sphère {
   public static void main(String [] args) {
      final float PI = 3.14159f;
      float radius,
            volume;
      try {
         radius = Float.parseFloat(
                    new BufferedReader(
                       new InputStreamReader(System.in)
                    ).readLine());
         volume = 4.0f/3.0f * PI * radius * radius * radius;
         System.out.println(volume);
      } catch (Exception e) {
      }
   }
}
```

Sequence, arithmetic and basic I/O

```
import java.io.*;
public class Sphère {
   public static void main(String [] args) {
      final float PI = 3.14159f;
      float radius,
            volume;
      try {
         radius = Flo



         volume = 4                                         ;
         System.out
      } catch (Excep
      }
   }
}
```

Constants and variables have to be declared (and defined) before use

Types appear (this is mistifying to many)

The ';' symbol

Sequence, arithmetic and basic I/O

```
import java.io.*;
public class Sphère {
   public static void main(String [] args) {
      final float PI = 3.14159f;
      float radius,
            volume;
         volume = 4.0f/3.0f * PI * radius * radius * radius;
         System.out.println(volume);
      } catch (Exception e) {
      }
   }
}
```

I typically write the literal without ‘f’ suffix first to get en error and discuss the fact that there are more than one floating point type. We read the message the compiler gave us, and they tell me what I should do

Sequence, arithmetic and basic I/O

```
import java.io.*;
public class Sphère {
   public static
      final floa
      float radi
            volu
      try {
         radius =
                    new        edReader(
                       new InputStreamReader(System.in)
                    ).readLine());
         volume = 4.0f/3.0f * PI * radius * radius * radius;
         System.out.println(volume);
      } catch (Exception e) {
      }
   }
}
```

Note the other floating point literals. I use the same technique as in C++ (4/3, then a test plan, then a discussion and a fix)

Sequen /O

There is the translation from pseudocode to Java, which involves syntax:

Read x becomes … very much involved!

Print x becomes System.out.println(x); (or System.out.print())

x ← y becomes x = y

```
import java.io.
public class Sp
   public stati
      final flo
      float rad
            volume;
      try {
         radius = Float.parseFloat(
                     new BufferedReader(
                        new InputStreamReader(System.in)
                     ).readLine());
         volume = 4.0f/3.0f * PI * radius * radius * radius;
         System.out.println(volume);
      } catch (Exception e) {
      }
   }
}
```

Sequence arithmetic and basic I/O

```
import java.io.*
public class Sphe
   public static
      final floa
      float radi
            volum
      try {
         radius = Float.parseFloat(
                     new BufferedReader(
                        new InputStreamReader(System.in)
                     ).readLine());
         volume = 4.0f/3.0f * PI * radius * radius * radius;
         System.out.println(volume);
      } catch (Exception e) {
      }
   }
}
```

Note these two subtelties:

- We have float and Float on screen, which leads to reasonable questions… but these are beginners!
- We have to introduce some exception handling scaffolding even though it makes no sense to them yet

Sequence, arithmetic and basic I/O

- These languages do not have an exponent operator
  - This is a nice moment to introduce the fact that mathematical functions exist, and how to use them
  - These functions have passing resemblance in terms of behavior to the functions they have been used to

Sequence, arithmetic and basic I/O

- These languages do not have an exponent operator
  - This is a nice moment to introduce the fact that mathematical functions exist, and how to use them
  - These functions have passing resemblance in terms of behavior to the functions they have been used to

Maths:
y=f(x)
y and f(x) are equivalent

Sequence, arithmetic and basic I/O

- These languages do not have an exponent operator
  - This is a nice moment to introduce the fact that mathematical functions exist, and how to use them
  - These functions have passing resemblance in terms of behavior to the functions they have been used to

Maths:
y=f(x)
y and f(x) are equivalent

In these languages:
y=f(x);
f does something with x, result of that computation is stored in y which can lead to a change of state

Sequence, arithmetic and basic I/O

```
#include <iostream>
#include <cmath>
using namespace std;
int main() {
   const float PI = 3.14159f;
   float radius,
         volume;
   cin >> radius;
   volume = 4.0f/3.0f * PI *
            pow(radius, 3);
   cout << volume << endl;
}
```

Sequence, arithmetic and basic I/O

```
#include <iostream>
#include <cmath>
using namespace std;
int main() {
   const float PI = 3.14159f;
   float radius,
         volume;
   cin >> radius;
   volume = 4.0f/3.0f * PI *
            pow(radius, 3);
   cout << volume << endl;
}
```

C++ has actual free functions, with overloads (very nice for beginners)

Sequence, arithmetic and basic I/O

```
using System;
namespace Sphère
{
   class Program
   {
      static void Main(string [] args)
      {
         const float PI = 3.14159f;
         float radius,
               volume;
         radius = float.Parse(Console.ReadLine());
         volume = 4.0f/3.0f * PI * (float)Math.Pow(radius,3);
         Console.WriteLine(volume);
      }
   }
}
```

Sequence, arithmetic and basic I/O

```
using System;
namespace Sphère
{
    class Program
    {
        static void Main(str
        {
            const float PI =
            float radius,
                  volume;
            radius = float.Parse(Console.ReadLine());
            volume = 4.0f/3.0f * PI * (float)Math.Pow(radius,3);
            Console.WriteLine(volume);
        }
    }
}
```

The Math.Pow() part is typically Ok, but the (necessary) cast from double to float is a difficulty for people who did not know what a type was a few hours before

Sequence, arithmetic and basic I/O

```
import java.io.*;
public class Sphère {
   public static void main(String [] args) {
      final float PI = 3.14159f;
      float radius,
            volume;
      try {
         radius = Float.parseFloat(
                    new BufferedReader(
                       new InputStreamReader(System.in)
                    ).readLine());
         volume = 4.0f/3.0f * PI * (float)Math.pow(radius, 3);
         System.out.println(volume);
      } catch (Exception e) {
      }
   }
}
```

Sequence, arithmetic and basic I/O

```
import java.io.*;
public class Sphère {
   public static void main(String [] args) {
      final float PI = 3.1415
      float radius,
            volume;
      try {
         radius = Float.parse
                    new BufferedReader(
                       new InputStreamReader(Syste   .in)
                    ).readLine());
         volume = 4.0f/3.0f * PI * (float)Math.pow(radius, 3);
         System.out.println(volume);
      } catch (Exception e) {
      }
   }
}
```

The Math.pow() part is typically Ok, but the (necessary) cast from double to float is a difficulty for people who did not know what a type was a few hours before

Calling Mathematical Functions

Calling Mathematical Functions

- Intent:
  - Solving a small problem with algebra yields familiarity, the need for a sequence of operations, and makes students combine operations in more complex ways
  - In what follows, they are to compute and display the perimeter of a triangle given its hypotenuse and one cathetus
  - It’s Pythagoras, but with a (small) twist to make them think about it

Calling Mathematical Functions

```
Read cathetusA
Read hypotenuse
cathetusB ← sqrt((hypotenuse^2 -
                  cathetusA^2))
perimeter ← hypotenuse +
            cathetusA +
            cathetusB
Print perimeter
```

Calling Mathematical Functions

```
Read cathetusA
Read hypotenuse
cathetusB ← sqrt((hypotenuse^2 -
                 cathetusA^2))
perimeter ← hypotenuse +


Print perimeter
```

In pseudocode, this is typically not an obstacle (the main point is how to express a square root)

Calling Mathematical Functions

```
import math
cathetusA = float(input("Known cathetus? "))
hypotenuse = float(input("Hypotenuse? "))
cathetusB = math.sqrt(pow(hypotenuse, 2) - pow(cathetusA,2))
perimeter = cathetusA + cathetusB + hypotenuse
print(perimeter)
```

Calling Mathematical Functions

```
import math
cathetusA = float(
cathetus? "))
hypotenuse = float(input("Hypotenuse?
"))
cathetusB = math.sqrt(pow(hypotenuse,
2) - pow(cathetusA,2))
perimeter = cathetusA + cathetusB +
hypotenuse
print(perimeter)
```

In Python, we need an import, but that can be explained quite nicely

Calling Mathematical Functions

```
import math
cathetusA = float(input("Known
cathetus? "))
hypotenuse = float(input("Hypotenuse?
"))
cathetusB = math.sq                     e,
2) - pow(cathetusA,
perimeter = cathetu
hypotenuse
print(perimeter)
```

Casts to float are a bit more work, but can be explained at this stage

Calling Mathematical Functions

```
import math
cathetusA = f
cathetus? "))
hypotenuse =                                    ?
"))
cathetusB = math.sqrt(pow(hypotenuse,
2) - pow(cathetusA,2))
perimeter = cathetusA + cathetusB +
hypotenuse
print(perimeter)
```

The actual computation translates quite directly from pseudocode, so we can concentrate on the mechanics of passing arguments and consuming the return value in order to do something with it

Calling Mathematical Functions

```
#include <iostream>
#include <cmath>
using namespace std;
int main() {
   float cathetusA,
         cathetusB,
         hypotenuse,
         perimeter;
   cin >> cathetusA;
   cin >> hypotenuse;
   cathetusB = sqrt(pow(hypotenuse,2) - pow(cathetusB,2));
   perimeter = hypotenuse + cathetusA +
               cathetusB;
   cout << perimeter << endl;
}
```

Calling Mathematical Functions

```
#include <iostream>
#include <cmath>
using namespace std;
int main() {
   float cathetusA,
         cathetusB,
         hypotenuse,
         perimeter;
   cin >> cathetusA;
   cin >> hypotenuse;
   cathetusB = sqrt(pow(hypotenuse,2)-
                    pow(cathetusA,2));
   perimeter = hypotenuse +
               cathetusA +
               cathetusB;
   cout << perimeter << endl;
}
```

In C++, we need an #incude, but that can be explained quite nicely

Calling Mathematical Functions

```
#include <iostream>
#include <cmath>
using namespace std;
int main() {
   float cathetusA,
         cathetusB,
         hypotenuse,
         perimeter;
   cin >> cathetusA;
   cin >> hypotenuse;
   cathetusB = sqrt(pow(hypotenuse,2) - pow(cathetusA,2));
   perimeter = hypotenuse + cathetusA +
               cathetusB;
   cout << perimeter << endl;
}
```

We are introducing a temporary variable (neither input nor output), and a function (sqrt())

We have a complex instruction with three functions interacting

Calling Mathematical Functions

```
using System;
namespace PérimètreTriangle
{
   class Program
   {
      static void Main(string [] args)
      {
         float cathetusA,
               cathetusB,
               hypotenuse,
               perimeter;
         cathetusA = float.Parse(Console.ReadLine());
         hypotenuse = float.Parse(Console.ReadLine());
         cathetusB = (float) (Math.Sqrt(Math.Pow(hypotenuse,2)-
                                       Math.Pow(cathetusA,2)));
         perimeter = hypotenuse +
                     cathetusA +
                     cathetusB;
         Console.WriteLine(perimeter);
      }
   }
}
```

Calling Mathematical Functions

```
using System;
namespace PérimètreTriangle
{
   class Program
   {
      static void Main(string []
      {
         float cathetusA,
               cathetusB,
               hypotenuse,
               perimeter;
         cathetusA = float.Parse(Console.ReadLine());
         hypotenuse = float.Parse(Console.ReadLine());
         cathetusB = (float) (Math.Sqrt(Math.Pow(hypotenuse,2)-
                                        Math.Pow(cathetusA,2)));
         perimeter = hypotenuse + cathetusA + cathetusB;
         Console.WriteLine(perimeter);
      }
   }
}
```

We are introducing a temporary variable (neither input nor output), and a function (Math.Sqrt())

We have a complex instruction with three functions interacting

Expressing the cast to float is difficult

Calling Mathematical Functions

```
import java.io.*;
public class PérimètreTriangle {
   public static void main(String [] args) {
      try {
         float cathetusA,
               cathetusB,
               hypotenuse,
               perimeter;
         cathetusA = Float.parseFloat(
            new BufferedReader(new InputStreamReader(System.in)).readLine()
         );
         hypotenuse = Float.parseFloat(
            new BufferedReader(new InputStreamReader(System.in)).readLine()
         );
         cathetusB = (float) Math.sqrt(Math.pow(hypotenuse,2)-
                                      Math.pow(cathetusA,2));
         perimeter = hypotenuse + cathetusA + cathetusB;
         System.out.println(perimeter);
      } catch (Exception e) {
      }
   }
}
```

Calling Mathematical Functions

```
import java.io.*;
public class PérimètreTriangle {
   public static void main(String [] arg
      try {
         float cathetusA,
               cathetusB,
               hypotenuse,
               perimeter;
         cathetusA = Float.parseFloat(
            new BufferedReader(new Input
         );
         hypotenuse = Float.parseFloat(
            new BufferedReader(new InputStreamRea           in)).readLine()
         );
         cathetusB = (float) Math.sqrt(Math.pow(hypotenuse,2)-
                                       Math.pow(cathetusA,2));
         perimeter = hypotenuse + cathetusA + cathetusB;
         System.out.println(perimeter);
      } catch (Exception e) {
      }
   }
}
```

We are introducing a temporary variable (neither input nor output), and a function (Math.sqrt())

We have a complex instruction with three functions interacting

Expressing the cast to float is difficult

Simple Decision-Making

Simple Decision-Making

- Intent:
  - We have people unused to logic outside of natural language constructs, where it's not always precise (to say the least)
  - There are many ideas involved with the humble if statement
    - Indentation
    - Syntax (in many languages, mandatory parentheses)
    - Braces (easy way out: impose them first)
    - When to put braces, when to use ';'
    - The role of the else statement

Simple Decision-Making

```
Read val
If val % 2 != 0
   Print "Odd"
Else
   Print "Even"
```

Simple Decision-Making

```
Read val
If val % 2 != 0
   Print "Odd"
Else
   Print "Even"
```

Most students have no idea what modulus means, and have forgotten integral division since elementary school. This is quite a shock, and takes time and practice to sink in

The != notation is unknown to most students at that stage

Simple Decision-Making

```
Read val
If val % 2 != 0
   Print "Odd"
Else
   Print "Even"
```

The simple logic behind if / else is trickier than we remember for beginners. Try to make a beginner write an algorithm to find the smallest of three numbers using if / else statements and relational operatores; many get it wrong

Simple Decision-Making

```
Read val
If val % 2 != 0
    Print "Odd"
Else
    Print "Even"
```

Indentation is essential in pseudocode. When students don't get it, I write

```
If A
    OpA
    If B
        OpB
    Else
        OpC
```

… and then I change (or remove) the indentation. It drives the point

Simple Decision-Making

```
number = int(input("Number? "))
if number % 2 != 0:
   print("Odd")
else:
   print("Even")
```

Simple Decision-Making

```
number = int(input("Number? "))
if number % 2 != 0:
   print("Odd")
else:
   print("Even")
```

Python syntax is close to pseudocode (except for the ':' symbol), and relies on indentation

The % and != operators are new syntax

Simple Decision-Making

```
#include <iostream>
using namespace std;
int main() {
   int val;
   cin >> val;
   if (val % 2 != 0) {
      cout << val << " is odd" << endl;
   } else {
      cout << val << " is even" << endl;
   }
}
```

Simple Decision-Making

```
#include <iostream>
using namespace std
int main() {
    int val;
    cin >> val;
    if (val % 2 != 0) {
        cout << val << " is odd" << endl;
    } else {
        cout << val << " is even" << endl;
    }
}
```

This is the first time we use an int variable. Distinguishing int and float can be an obstacle

Simple Decision-Making

```
#include <iostream>
using namespace std
int main() {
   int val;
   cin >> val;
   if (val % 2 != 0) {
      cout << val << " is odd" << endl;
   } else {
      cout << val << " is even" << endl;
   }
}
```

The modulus operator is new to them. Introducing != means it's an interesting moment to introduce them all (==, !=, <, <=, > and >=). Note the difficulty of distinguishing = and ==

Simple Decision-Making

```
#include <iostream>
using namespace std
int main() {
   int val;
   cin >> val;
   if (val % 2 != 0) {
      cout << val << " is odd" << endl;
   } else {
      cout << val << " is even" << endl;
   }
}
```

The if and else syntax is a lot to take in (parentheses, braces, ‘;’, indentation)

Simple Decision-Making

```
#include <iostream>
using namespace std;
int main() {
   int val;
   cin >> val;
   if (val % 2 != 0) {
      cout << val << " is odd" << endl;
   } else {
      cout << val << " is even" << endl;
   }
}
```

Chaining output is something new to learn

Simple Decision-Making

```
using System;
namespace Impairs
{
   class Program
   {
      static void Main(string [] args)
      {
         int val;
         val = int.Parse(Console.ReadLine());
         if (val %2 != 0)
         {
            Console.WriteLine("{0} is odd", val);
         }
         else
         {
            Console.WriteLine("{0} is even", val);
         }
      }
   }
}
```

Simple Decision-Making

```
using System;
namespace Impairs
{
   class Program
   {
      static void Main(string [] ar
      {
         int val;
         val = int.Parse(Console.ReadLine());
         if (val %2 != 0)
         {
            Console.WriteLine("{0} is odd", val);
         }
         else
         {
            Console.WriteLine("{0} is even", val);
         }
      }
   }
}
```

This is the first time we use an int variable. Distinguishing int and float can be an obstacle. Note that reading an int requires different syntax from reading a float

Simple Decision-Making

```
using System;
namespace Impairs
{
   class Program
   {
      static void Main(string [] ar
      {
         int val;
         val = int.Parse(Console.ReadLine());
         if (val %2 != 0)
         {
            Console.WriteLine("{0} is odd", val);
         }
         else
         {
            Console.WriteLine("{0} is even", val);
         }
      }
   }
}
```

The modulus operator is new to them. Introducing != means it’s an interesting moment to introduce them all (==, !=, <, <=, > and >=). Note the difficulty of distinguishing = and ==

Simple Decision-Making

```
using System;
namespace Impairs
{
   class Program
   {
      static void Main(string [] args
      {
         int val;
         val = int.Parse(Console.ReadLine());
         if (val %2 != 0)
         {
            Console.WriteLine("{0} is odd", val);
         }
         else
         {
            Console.WriteLine("{0} is even", val);
         }
      }
   }
}
```

The if and else syntax is a lot to take in (parentheses, braces, ‘;’, indentation)

Simple Decision-Making

```
using System;
namespace Impairs
{
   class Program
   {
      static void Main(string [
      {
         int val;
         val = int.Parse(Console
         if (val %2 != 0)
         {
            Console.WriteLine("{0} is odd", val);
         }
         else
         {
            Console.WriteLine("{0} is even", val);
         }
      }
   }
}
```

Complex output requires syntax and a zero-based numbering, for people unused to function at this point

Simple Decision-Making

```
import java.io.*;
public class Impairs {
   public static void main(String [] args) {
      try {
         int val;
         val = Integer.parseInt(
                    new BufferedReader(
                       new InputStreamReader(System.in)
                    ).readLine());
         if (val % 2 != 0) {
            System.out.println(val + " is odd");
         } else {
            System.out.println(val + " is even");
         }
      } catch (Exception e) {
      }
   }
}
```

Simple Decision-Making

```
import java.io.*;
public class Impairs {
   public static void main(String []
      try {
         int val;
         val = Integer.parseInt(
                    new BufferedReader(
                       new InputStreamReader(System.in)
                    ).readLine());
         if (val % 2 != 0) {
            System.out.println(val + " is odd");
         } else {
            System.out.println(val + " is even");
         }
      } catch (Exception e) {
      }
   }
}
```

This is the first time we use an int variable. Distinguishing int and float can be an obstacle. Note that reading an int requires different syntax from reading a float, and introduces Integer

Simple Decision-Making

```
import java.io.*;
public class Impairs {
   public static void main(String
      try {
         int val;
         val = Integer.parseI
                    new BufferedRe
                       new InputStreamReader(System.in)
                    ).readLine());
         if (val % 2 != 0) {
            System.out.println(val + " is odd");
         } else {
            System.out.println(val + " is even");
         }
      } catch (Exception e) {
      }
   }
}
```

The modulus operator is new to them. Introducing != means it's an interesting moment to introduce them all (==, !=, <, <=, > and >=). Note the difficulty of distinguishing = and ==

Simple Decision-Making

```
import java.io.*;
public class Impairs {
   public static void main(String
      try {
         int val;
         val = Integer.parse nt(
                    new BufferedReader(
                       new InputStreamReader(System.in)
                    ).readLine());
         if (val % 2 != 0) {
            System.out.println(val + " is odd");
         } else {
            System.out.println(val + " is even");
         }
      } catch (Exception e) {
      }
   }
}
```

The if and else syntax is a lot to take in (parentheses, braces, ';', indentation)

Simple Decision-Making

```
import java.io.*;
public class Impairs {
   public static void main(String [] args) {
      try {
         int val;
         val = Integer.pars
                    new Bu
                      new
                    ).readLine())
         if (val % 2 != 0) {
            System.out.println(val + " is odd");
         } else {
            System.out.println(val + " is even");
         }
      } catch (Exception e) {
      }
   }
}
```

Complex output involves concatenating text with other things, which is a new idea at this stage

Writing a Function

Writing a Function

- Intent:
  - Using function is important
  - Writing a function completes the picture
  - There is a pathway from arguments at the call site, to arguments in the function, to return value, to return value consumption that is quite complex
    - Concepts such as scope, local variable, naming all become more complex
  - With beginners, the concept of scoped names is surprising and challenging at first
  - The concept of positional (instead of named) arguments takes some getting used to
    - Static typing is helpful!

Writing a Function

```
Function Sum(a, b)
    res ← a + b
    Return res
Function Main
    x ← 2
    y ← 3
    result ← Sum(x,y)
    Print result
```

Writing a Function

```
Function Sum(a, b)
    res ← a + b
    Return res
Function Main
    x ← 2
    y ← 3
    result ← Sum(x,y)
    Print result
```

Writing a Function

```
Function Sum(a, b)
    res ← a + b
    Return res
Function Main
    x ← 2
    y ← 3
    result ← Sum(x,y)
    Print result
```

Writing a Function

```
def Sum(a, b):
    res = a + b
    return res
x = 2
y = 3
result = Sum(x,y)
print(result)
```

Writing a Function

```
def Sum(a, b):
    res = a + b
    return res
x = 2
y = 3
result = Sum(x,y)
print(result)
```

Writing a Function

```
def Sum(a, b):
    res = a + b
    return res
x = 2
y = 3
result = Sum(x,y)
print(result)
```

Writing a Function

```
#include <iostream>
using namespace std;
int sum(int a, int b) {
   int res;
   res = a + b;
   return res;
}
int main() {
   int x, y, result;
   x = 2;
   y = 3;
   result = sum(x, y);
   cout << result;
}
```

Writing a Function

```
#include <iostream>
using namespace std;
int sum(int a, int b) {
   int res;
   res = a + b;
   return res;
}
int main() {
   int x, y, result;
   x = 2;
   y = 3;
   result = sum(x, y);
   cout << result;
}
```

Writing a Function

```
#include <iostream>
using namespace std;
int sum(int a, int b) {
   int res;
   res = a + b;
   return res;
}
int main() {
   int x, y, result;
   x = 2;
   y = 3;
   result = sum(x, y);
   cout << result;
}
```

Writing a Function

```
#include <iostream>
using namespace std;
int sum(int a, int b) {
   int res;
   res = a + b;
   return res;
}
int main() {
   int x, y, result;
   x = 2;
   y = 3;
   result = sum(x, y);
   cout << result;
}
```

Writing a Function

```
using System;
namespace Function
{
   class Program
   {
      static int Sum(int a, int b)
      {
         int res;
         res = a + b;
         return res;
      }
      static void Main()
      {
         int x, y, result;
         x = 2;
         y = 3;
         result = Sum(x, y);
         Console.WriteLine(result);
      }
   }
}
```

Writing a Function

```
using System;
namespace Function
{
   class Program
   {
      static int Sum(int a, int b)
      {
         int res;
         res = a + b;
         return res;
      }
      static void Main()
      {
         int x, y, result;
         x = 2;
         y = 3;
         result = Sum(x, y);
         Console.WriteLine(result);
      }
   }
}
```

Writing a Function

```
using System;
namespace Function
{
   class Program
   {
      static int Sum(int a, int b)
      {
         int res;
         res = a + b;
         return res;
      }
      static void Main()
      {
         int x, y, result;
         x = 2;
         y = 3;
         result = Sum(x, y);
         Console.WriteLine(result);
      }
   }
}
```

Writing a Function

```
using System;
namespace Function
{
   class Program
   {
      static int Sum(int a, int b)
      {
         int res;
         res = a + b;
         return res;
      }
      static void Main()
      {
         int x, y, result;
         x = 2;
         y = 3;
         result = Sum(x, y);
         Console.WriteLine(result);
      }
   }
}
```

Writing a Function

```
using System;
namespace Function
{
   class Program
   {
      static int Sum(int a, int b)
      {
         int res;
         res = a + b;
         return res;
      }
      static void Main()
      {
         int x, y, result;
         x = 2;
         y = 3;
         result = Sum(x, y);
         Console.WriteLine(result);
      }
   }
}
```

Writing a Function

```
import java.io.*;
class Function {
   static int sum(int a, int b) {
      int res;
      res = a + b;
      return res;
   }
   public static void main(String [] args) {
      int x, y, result;
      x = 2;
      y = 3;
      result = sum(x, y);
      System.out.println(result);
   }
}
```

Writing a Function

```
import java.io.*;
class Function {
   static int sum(int a, int b) {
      int res;
      res = a + b;
      return res;
   }
   public static void main(String [] args) {
      int x, y, result;
      x = 2;
      y = 3;
      result = sum(x, y);
      System.out.println(result);
   }
}
```

Things to notice are the same as with C#, really

Managing Complexity

Managing Complexity

- There are many difficult things inherent to learning programming
  - We take them for granted, we forget
  - In most of our "main", commercial languages, these difficulties are conceptual
    - Syntax can help or hinder, but that basic ideas are not that dissimilar
    - Some things (e.g. keyboard input in Java) are more painful, obviously, but one could hide this somewhat
  - There are many things one needs to learn in order to program
    - Most of these are not "language" things, at least not at first

Managing Complexity

- Teaching is not an "I'm so cool" activity
  - We are not there to showcase knowledge
  - We are there because we want to guide and accompany those who are learning
- There have been no complex language features in these examples
  - Show what is necessary to support your message
  - Try to get the idea across, keep only necessary complexity

Managing Complexity

- There are many things one can do to help students learn to program in C++
  - Prefer vector to raw arrays
    - They are simpler to pass to functions, return from functions
  - Prefer string to nul-delimited char sequences
    - We have "Yo"s literals now, why not use them?
  - Prefer references to pointers
    - The low-level things like pointers and raw arrays are important to C++, but not required for beginners to learn to program in that language

Missing Pieces and Pain Points

Missing Pieces and Pain Points

- In our beginner classes, we of course do many, many more exercices and assignments
  - They code every week, and bring back assignments that we read and grade and (constructively) criticize
  - We use relational operators when introducing if statements
  - We use logic operators when introducing loops
  - We take our time. By mid-semester, they write multi-function programs and we start giving them more leeway in their “architectural” choices

Missing Pieces and Pain Points

- By the end of the first semester, they can do relatively complex programs
  - A program that manages a simple pet shop
  - A labyrinth with monsters moving randomly that a hero has to get out of
  - A war between vampires and werewolves fighting one another
- None of this requires low-level things
- The levels of difficulty involved are pretty much the same from language to language

Missing Pieces and Pain Points

- Some things are not helping C++ with beginners
  - Some standard GUI tools would be welcome
  - Basic I/O is quite different from language to language

Missing Pieces and Pain Points

- Difficulties exist everywhere
- Programming is difficult
- Programming is fun

Postcondition

Postcondition

- This talk is based on actual experience

Questions?