PHY410 / 505
Computational Physics 1

Salvatore Rappoccio

Computational Physics

- Welcome to Computational Physics 1!

- This is the first half of a 2-semester sequence intended as an intro to numerical computing and computational physics

- The class will be hybrid in-class / video-based lecture approach
- There will be some students from other universities and from Fermi National Accelerator Laboratory
  - Welcome!

Computational Physics

- I assume you are:
  - At least a mid-level undergraduate
  - Interested in computational techniques

- I assume you know:
  - CS preqrequisites:
    - CSE >=113 or equivalent:
    - 1 semester of programming something
    - I assume you know >=1 programming language
  - Math prerequisites:
    - Calculus (2 semesters), Vector calculus (1 semester)
    - Ordinary differential equations (1 semester)
  - Physics prerequisites:
    - Introductory mechanics
    - Introductory E+M
    - Introductory modern physics / quantum mechanics

Computational Physics

- Due to a lack of students having sufficient computational skills, will cover technical computing first

- The first 4-8 weeks will focus on technical computing aspects :
  - UNIX environments
  - Compiling
  - Debugging
  - Programming in C++ and python
  - Version control (git)
  - Containers (docker)
  - Algorithmic control
  - Vectorization
  - C++ and python together

Computational Physics

- Subsequently, we will move into the actual computational physics part:

  - Simple data analysis
  - Numerical algorithms
  - Linear algebra
  - Nonlinear equations
  - Ordinary differential equations
  - Statistical and simulation methods

- Semester 2 will pick up from there!
  - Includes introduction to machine learning, so worth doing!

Computational Physics

- Class expectations:
  - I don't take attendance. You're adults, and are responsible.
  - Politeness and appropriate conduct (never had a problem with this, but still…)

- Workload:
  - This course is **hands-on** and **intensive**
  - Expectation is ~10 hours / week including 3 hours class time
  - You will learn a LOT from homework and exams
  - You will gain fairly little by just coming to class and not participating in assignments

UBLearns and Syllabus

- For those registered at UB: the class is organized via the "Blackboard" tool UBLearns

ALL OFFICIAL UB COMMUNICATION
WILL BE THROUGH UBLEARNS.
REGISTER IMMEDIATELY

For those auditing class: I will forward communication to you via indico.

Hands-on Sessions

- This class has part lecture and part hands-on sessions
  - Utilize them! Ask questions, get together with friends and code things, try crazy ideas, etc

- You are expected to come to all lectures but again, I don’t take attendance.

Non-UB Students : Caveats

- There are 3x as many of you as there are UB students!

- This is basically "freemium". You will have part of the content but I cannot provide certain functions without a UB account (which requires registration).
  - No grading
  - No hands-on session support
  - Best-effort answering questions

If you want to upgrade to "Premium" register here!

https://registrar.buffalo.edu/nondegree

C++ / Python

- We'll be using C++ and python
  - python :
    - interpreted
    - quick to develop : it's practically pseudocode
    - slow to execute : but this can be mitigated with numpy!
    - very simple
    - good for scripting and string handling
  - C++ :
    - compiled
    - slow to develop
    - fast to execute
    - very robust
    - good for high-performance computing

- Often scientists develop prototypes with python and then convert them to C++, although numpy helps a lot
  - Will ALSO cover how to use C++ from within python, so you get the best of both worlds!

C++ / Python

- Why not `matlab`?
  - Not everyone has access to it, it's not free, and so I don't want to make you use it
  - No examples will be provided there

- Why not Fortran90?
  - It's a very, very, very old language
    - F77 is older than me, but just barely
    - F90 is older than most (all?) of you
  - It's very useful for scientific computing, but it's really losing its support base except for sci. comp.
  - I won't make you learn it since it has limited utility outside of academia and you probably will never see it anywhere else

Numerical Methods for Physics

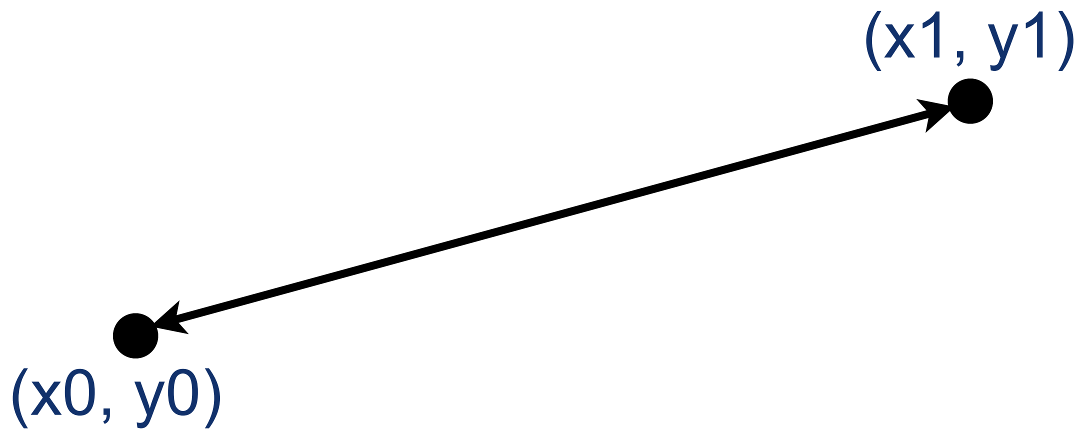
- The following textbooks are required (and free of charge). You are expected to have working knowledge of things covered in these books.
  - Fundamentals of C++ Programming by Richard Halterman
    - Example code at https://github.com/halterman/CppBook-SourceCode
  - https://www.tutorialspoint.com/python3/ : Introduction to python •
  - Numerical Recipes in C++ :
    - The latest version does cost money but is a worthwhile investment for your career, while older versions of NR are free. Earlier online version of NR for free

Development

- Step 1 : Write the algorithm down on paper
- Step 2 :
  - If you don’t understand everything : goto step 1
  - else : continue
- Step 3 : write pseudocode
- Step 4 : continue
  - If you don’t understand everything : goto step 3
  - else : continue
- Step 5 : write code
- Step 6 : check code with unit tests
  - Check “pass” criterion
  - Check “fail” criterion
  - If unit test fails : goto Step 5
  - else : continue
- Step 7 : Publish!

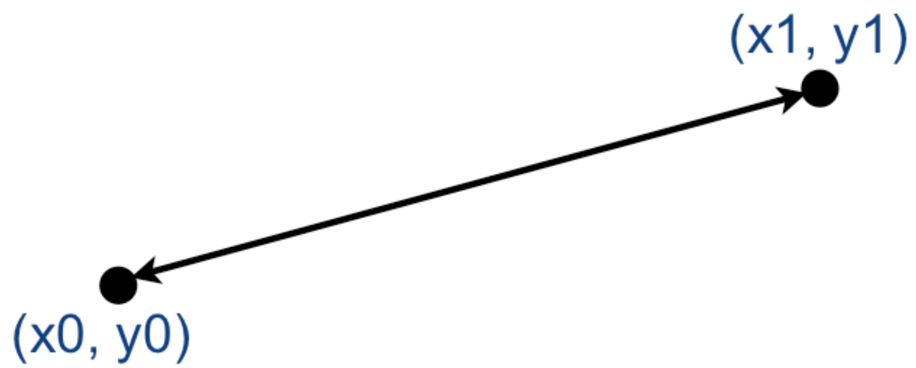
Development

- Example : Compute the slope between two points



Development

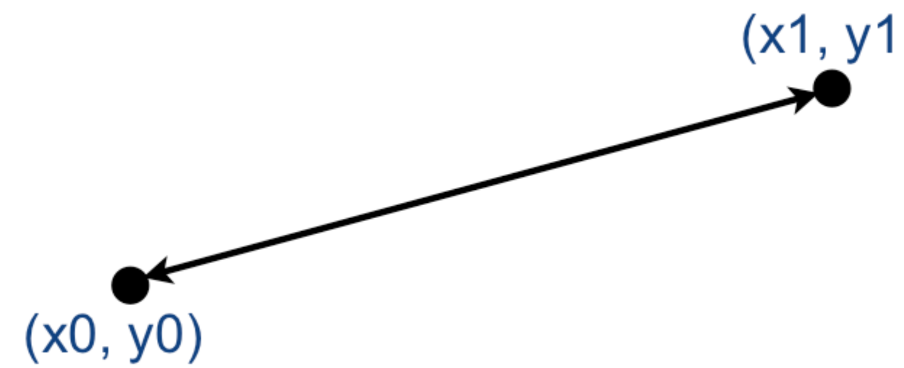
- Step 1 : Write the algorithm down on paper
- Step 2 :
  - If you don't understand everything : goto step 1
  - else : continue
- Step 3 : write pseudocode
- Step 4 : continue
  - If you don't understand everything : goto step 3
  - else : continue
- Step 5 : write code
- Step 6 : check code with unit tests
  - Check "pass" criterion
  - Check "fail" criterion
  - If unit test fails : goto Step 5
  - else : continue
- Step 7 : Publish!



$$slope = (y1 - y0) / (x1 - x0)$$

Development

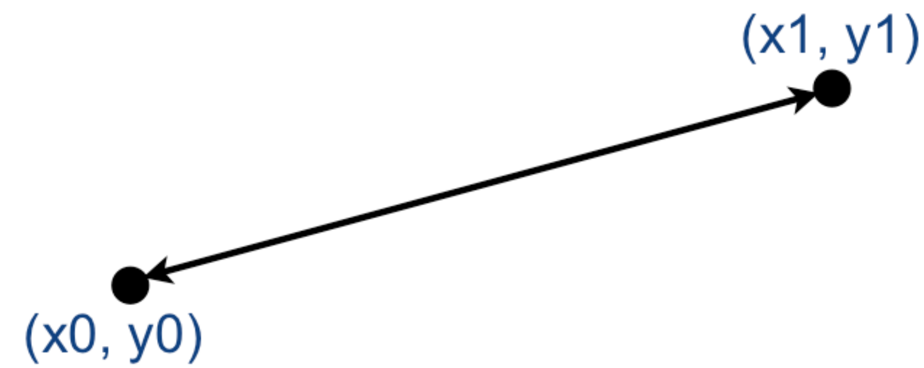
- Step 1 : Write the algorithm down on paper
- Step 2 :
  - If you don't understand everything : goto step 1
  - else : continue
- Step 3 : write pseudocode
- Step 4 : continue
  - If you don't understand everything : goto step 3
  - else : continue
- Step 5 : write code
- Step 6 : check code with unit tests
  - Check "pass" criterion
  - Check "fail" criterion
  - If unit test fails : goto Step 5
  - else : continue
- Step 7 : Publish!



$$slope = (y1 - y0) / (x1 - x0)$$

Development

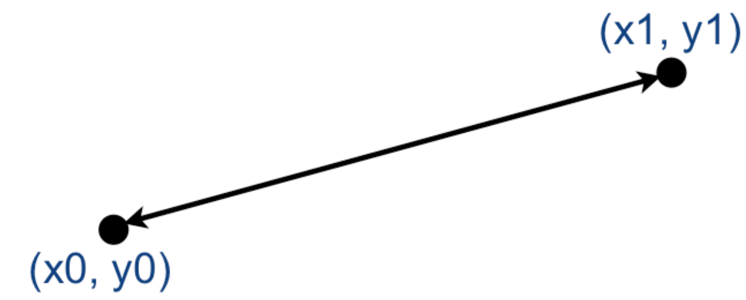
- Step 1 : Write the algorithm down on paper
- Step 2 :
  - If you don't understand everything : goto step 1
  - else : continue
- Step 3 : write pseudocode
- Step 4 : continue
  - If you don't understand everything : goto step 3
  - else : continue
- Step 5 : write code
- Step 6 : check code with unit tests
  - Check "pass" criterion
  - Check "fail" criterion
  - If unit test fails : goto Step 5
  - else : continue
- Step 7 : Publish!



$$slope = (y1 - y0) / (x1 - x0)$$

Development

- Step 1 : Write the algorithm down on paper
- Step 2 :
  - If you don’t understand everything : goto step 1
  - else : continue
- Step 3 : write pseudocode
- Step 4 : continue
  - If you don’t understand everything : goto step 3
  - else : continue
- Step 5 : write code
- Step 6 : check code with unit tests
  - Check “pass” criterion
  - Check “fail” criterion
  - If unit test fails : goto Step 5
  - else : continue
- Step 7 : Publish!

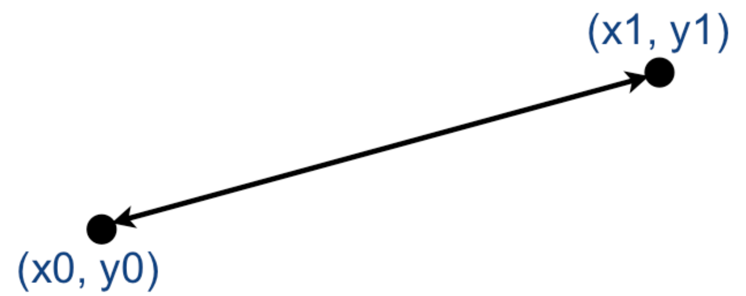


slope = (y1 - y0) / (x1 - x0)

```
slope = (y1-y0) / (x1-x0)
return slope
```

Development

- Step 1 : Write the algorithm down on paper
- Step 2 :
  - If you don't understand everything : goto step 1
  - else : continue
- Step 3 : write pseudocode
- Step 4 : continue
  - If you don't understand everything : goto step 3
  - else : continue
- Step 5 : write code
- Step 6 : check code with unit tests
  - Check "pass" criterion
  - Check "fail" criterion
  - If unit test fails : goto Step 5
  - else : continue
- Step 7 : Publish!

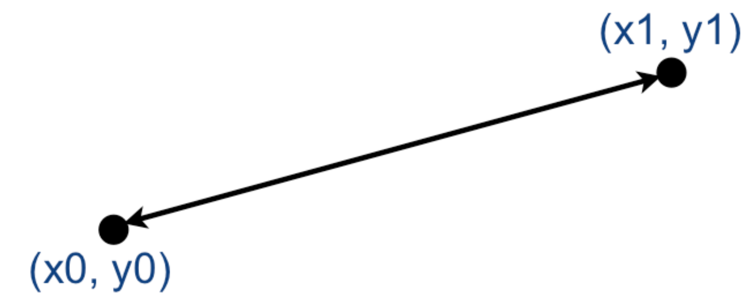


slope = (y1 - y0) / (x1 - x0)

```
slope = (y1-y0) / (x1-x0)
return slope
```

Development

- Step 1 : Write the algorithm down on paper
- Step 2 :
  - If you don’t understand everything : goto step 1
  - else : continue
- Step 3 : write pseudocode
- Step 4 : continue
  - If you don’t understand everything : goto step 3
  - else : continue
- Step 5 : write code
- Step 6 : check code with unit tests
  - Check “pass” criterion
  - Check “fail” criterion
  - If unit test fails : goto Step 5
  - else : continue
- Step 7 : Publish!

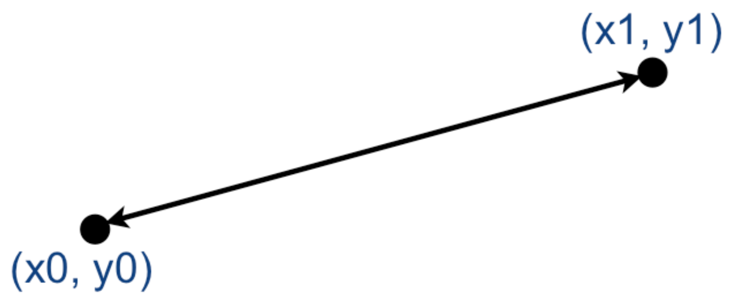


slope = (y1 - y0) / (x1 - x0)

```
slope = (y1-y0) / (x1-x0)
return slope
```

Development

- Step 1 : Write the algorithm down on paper
- Step 2 :
  - If you don't understand everything : goto step 1
  - else : continue
- Step 3 : write pseudocode
- Step 4 : continue
  - If you don't understand everything : goto step 3
  - else : continue
- Step 5 : write code
- Step 6 : check code with unit tests
  - Check "pass" criterion
  - Check "fail" criterion
  - If unit test fails : goto Step 5
  - else : continue
- Step 7 : Publish!

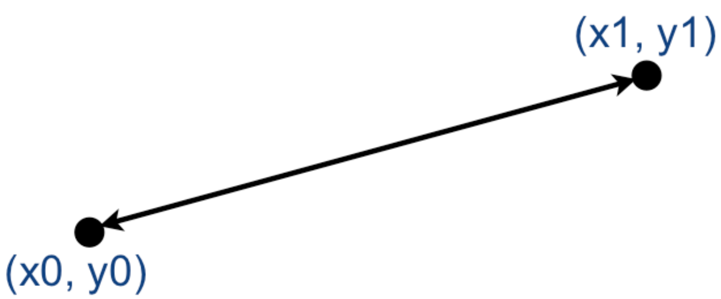


slope = (y1 - y0) / (x1 - x0)

```
slope = (y1-y0) / (x1-x0)
return slope
```

```
def getSlope ( x0,y0, x1, y1) :
    slope = (y1-y0) / (x1-x0)
    return slope
```

Development



slope = (y1 - y0) / (x1 - x0)

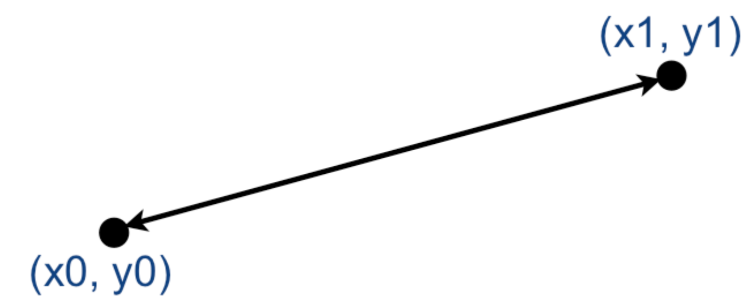
- Step 1 : Write the algorithm down on paper
- Step 2 :
  - If you don't understand everything : goto step 1
  - else : continue
- Step 3 : write pseudocode
- Step 4 : continue
  - If you don't understand everything : goto step 3
  - else : continue
- Step 5 : write code
- Step 6 : check code with unit tests
  - Check "pass" criterion
  - Check "fail" criterion
  - If unit test fails : goto Step 5
  - else : continue
- Step 7 : Publish!

```
slope = (y1-y0) / (x1-x0)
return slope
```

```
from math import *

def getSlope ( x0,y0, x1, y1) :
    slope = (y1-y0) / (x1-x0)
    return slope


#unit tests!
print 'Unit test 1'
x0 = 0.0
y0 = 0.0
x1 = 1.0
y1 = 1.0
slope = getSlope( x0,y0,x1,y1 )
print slope
```

Development

- Step 1 : Write the algorithm down on paper
- Step 2 :
  - If you don’t understand everything : goto step 1
  - else : continue
- Step 3 : write pseudocode
- Step 4 : continue
  - If you don’t understand everything : goto step 3
  - else : continue
- Step 5 : write code
- Step 6 : check code with unit tests
  - Check “pass” criterion
  - Check “fail” criterion
  - If unit test fails : goto Step 5
  - else : continue
- Step 7 : Publish!



slope = (y1 - y0) / (x1 - x0)

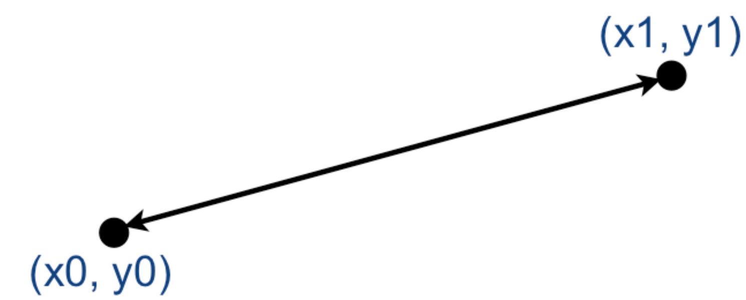
```
slope = (y1-y0) / (x1-x0)
return slope
```

```
from math import *

def getSlope ( x0,y0, x1, y1) :
    slope = (y1-y0) / (x1-x0)
    return slope


#unit tests!
y0 = 0.0
x1 = 1.0
y1 = 1.0
slope = getSlope( x0,y0,x1,y1 )
print slope
```

```
Unit test 1
1.0
```

OK!

Development



slope = (y1 - y0) / (x1 - x0)

```
slope = (y1-y0) / (x1-x0)
return slope
```

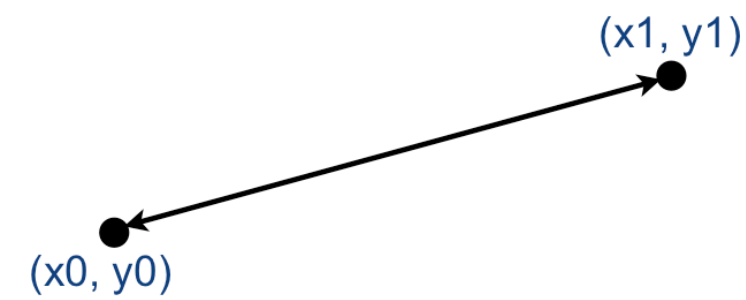
- Step 1 : Write the algorithm down on paper
- Step 2 :
  - If you don't understand everything : goto step 1
  - else : continue
- Step 3 : write pseudocode
- Step 4 : continue
  - If you don't understand everything : goto step 3
  - else : continue
- Step 5 : write code
- Step 6 : check code with unit tests
  - Check "pass" criterion
  - Check "fail" criterion
  - If unit test fails : goto Step 5
  - else : continue
- Step 7 : Publish!

```
from math import *

def getSlope ( x0,y0, x1, y1) :
    slope = (y1-y0) / (x1-x0)
    return slope


#unit tests!
print 'Unit test 1'
x0 = 0.0
y0 = 0.0
x1 = 0.0
y1 = 1.0
slope = getSlope( x0,y0,x1,y1 )
print slope
```

Hmmmm....

Development

- Step 1 : Write the algorithm down on paper
- Step 2 :
  - If you don’t understand everything : goto step 1
  - else : continue
- Step 3 : write pseudocode
- Step 4 : continue
  - If you don’t understand everyt
  - else : continue
- Step 5 : write code
- Step 6 : check code with unit test
  - Check “pass” criterion
  - Check “fail” criterion
  - If unit test fails : goto Step 5
  - else : continue
- Step 7 : Publish!



slope = (y1 - y0) / (x1 - x0)

```
slope = (y1-y0) / (x1-x0)
return slope
```

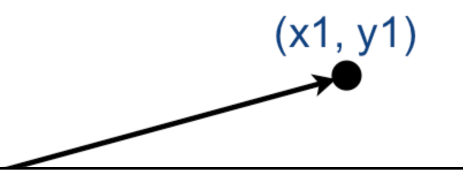
```
Unit test 1
1.0
Unit test 2
Traceback (most recent call last):
  File "slope.py", line 24, in <module>
    slope = getSlope( x0,y0,x1,y1 )
  File "slope.py", line 4, in getSlope
    slope = (y1-y0) / (x1-x0)
ZeroDivisionError: float division by zero
```

Not a good handling!

```
y0 = 0.0
x1 = 0.0
y1 = 1.0
slope = getSlope( x0,y0,x1,y1 )
print slope
```

Hmmmm....

Development



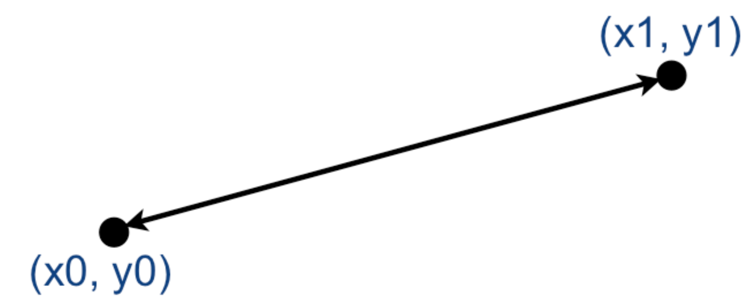
- Step 1 : Write the algorith
- Step 2 :
  - If you don't understan
  - else : continue
- Step 3 : write pseudocod
- Step 4 : continue
  - If you don't understan
  - else : continue
- Step 5 : write code
- Step 6 : check code with
  - Check "pass" criterion
  - Check "fail" criterion
  - If unit test fails : goto S
  - else : continue
- Step 7 : Publish!

```
from math import *


def getSlope ( x0,y0, x1, y1) :
    num = y1-y0
    den = x1-x0
    if abs(den) > 0.00001 :
        slope = num / den
    else :
        print 'invalid input, setting to None'
        slope = None
    return slope


#unit tests!
print 'Unit test 1'
x0 = 0.0
y0 = 0.0
x1 = 0.0
y1 = 1.0
slope = getSlope( x0,y0,x1,y1 )
print slope
```

Better! Checks within numerical precision, throws an error condition when something went wrong!

Development

- Step 1 : Write the algorithm down on paper
- Step 2 :
  - If you don’t understand everything : goto step 1
  - else : continue
- Step 3 : write pseudocode
- Step 4 : continue
  - If you don’t understand everyt
  - else : continue
- Step 5 : write code
- Step 6 : check code with unit test
  - Check “pass” criterion
  - Check “fail” criterion
  - If unit test fails : goto Step 5
  - else : continue
- Step 7 : Publish!



slope = (y1 - y0) / (x1 - x0)

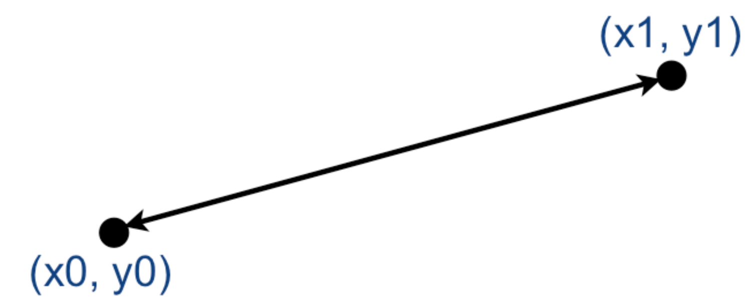
```
slope = (y1-y0) / (x1-x0)
return slope
```

```
Unit test 1
1.0
Unit test 2
----> invalid input, setting to None
None
```

```
        slope = None
    return slope

#unit tests!
print 'Unit test 1'
x0 = 0.0
y0 = 0.0
x1 = 0.0
y1 = 1.0
slope = getSlope( x0,y0,x1,y1 )
print slope
```

Development

- Step 1 : Write the algorithm down on paper
- Step 2 :
  - If you don't understand everything : goto step 1
  - else : continue
- Step 3 : write pseudocode
- Step 4 : continue
  - If you don't understand everything : goto step 3
  - else : continue
- Step 5 : write code
- Step 6 : check code with unit tests
  - Check "pass" criterion
  - Check "fail" criterion
  - If unit test fails : goto Step 5
  - else : continue
- Step 7 : Publish!



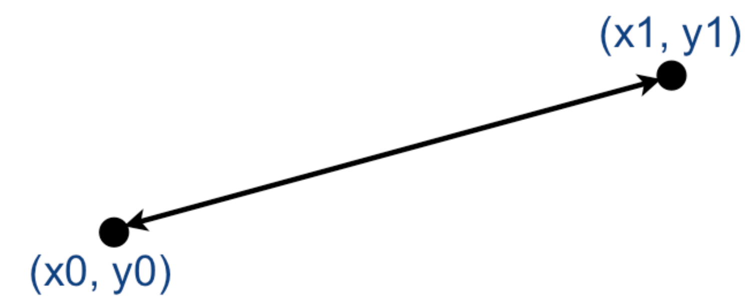
slope = (y1 - y0) / (x1 - x0)

```
slope = (y1-y0) / (x1-x0)
return slope
```

```
from math import *


def getSlope ( x0,y0, x1, y1) :
    num = y1-y0
    den = x1-x0
    if abs(den) > 0.00001 :
        slope = num / den
    else :
        print 'invalid input, setting to None'
        slope = None
    return slope


#unit tests!
print 'Unit test 1'
x0 = 0.0
y0 = 0.0
x1 = 0.0
y1 = 1.0
slope = getSlope( x0,y0,x1,y1 )
print slope
```

Development

- Step 1 : Write the algorithm down on paper
- Step 2 :
  - If you don't understand everything : goto step 1
  - else : continue
- Step 3 : write pseudocode
- Step 4 : continue
  - If you don't understand everything : goto step 3
  - else : continue
- Step 5 : write code
- Step 6 : check code with unit tests
  - Check "pass" criterion
  - Check "fail" criterion
  - If unit test fails : goto Step 5
  - else : continue
- Step 7 : Publish!



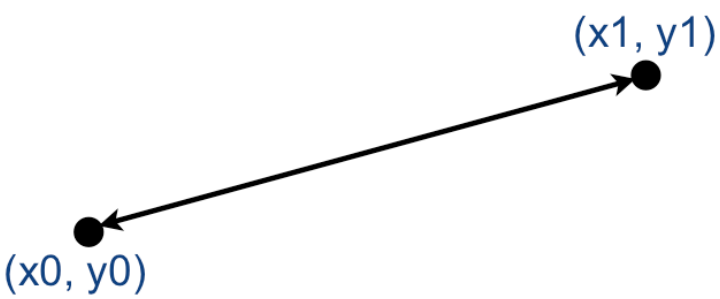
slope = (y1 - y0) / (x1 - x0)

```
slope = (y1-y0) / (x1-x0)
return slope
```

```
from math import *


def getSlope ( x0,y0, x1, y1) :
    num = y1-y0
    den = x1-x0
    if abs(den) > 0.00001 :
        slope = num / den
    else :
        print 'invalid input, setting to None'
        slope = None
    return slope


#unit tests!
print 'Unit test 1'
x0 = 0.0
y0 = 0.0
x1 = 0.0
y1 = 1.0
slope = getSlope( x0,y0,x1,y1 )
print slope
```

Development

- Step 1 : Write the algorithm down on paper
- Step 2 :
  - If you don’t understand everything : goto step 1
  - else : continue
- Step 3 : write pseudocode
- Step 4 : continue
  - If you don’t understand everything : goto step 3
  - else : continue
- Step 5 : write code
- Step 6 : check code with unit tests
  - Check “pass” criterion
  - Check “fail” criterion
  - If unit test fails : goto Step 5
  - else : continue
- Step 7 : Publish!



slope = (y1 - y0) / (x1 - x0)

```
slope = (y1-y0) / (x1-x0)
return slope
```

```
from math import *


def getSlope ( x0,y0, x1, y1) :
    num = y1-y0
    den = x1-x0
    if abs(den) > 0.00001 :
        slope = num / den
    else :
        print 'invalid input, setting to None'
        slope = None
    return slope


#unit tests!
print 'Unit test 1'
x0 = 0.0
y0 = 0.0
x1 = 0.0
y1 = 1.0
slope = getSlope( x0,y0,x1,y1 )
print slope
```

To UBLearns!

Development

- A few “take-home” messages :
  - Machines do not understand infinite precision
    - We’ll get to that
  - Machines do exactly what we ask them to
    - We’ll get to that too
  - Thinking ahead can save you a lot of time!

- We’ve seen the power of python
  - Looks almost like pseudocode (which you should be writing anyway)
  - Good as a first stab at something
  - But, very slow loops!

Development

- Next, let's translate this to C++

```
#include <iostream>
#include <math.h>

using std::cout;
using std::endl;

// Better slope calculator : Error handling when
// denominator is zero
float getSlope( float x0, float y0, float x1, float y1) {
  float num = y1 - y0;
  float den = x1 - x0;
  float slope = 0.0;
  if ( fabs(den) > 0.00001 ) {
    slope = num / den;
  } else {
    cout << "----> invalid input, setting to -9999" << endl;
    slope = -9999;
  }
  return slope;
}
```

```
int main( void )
{
  // Unit tests!

  cout << "Unit test 1" << endl;
  float x0 = 0.0;
  float y0 = 0.0;
  float x1 = 1.0;
  float y1 = 1.0;
  float slope = getSlope( x0,y0,x1,y1 );
  cout << slope << endl;

  cout << "Unit test 2" << endl;
  x0 = 0.0;
  y0 = 0.0;
  x1 = 0.0;
  y1 = 1.0;
  slope = getSlope( x0,y0,x1,y1 );
  cout << slope << endl;

  return 0;
}
```

Let's get started!

- This should wet your whistle and your appetite, I hope.
- Wednesday we dive right into computing and the software requirements