

# Introduction to Programming with Scientific Applications

Gerth Stølting Brodal  
Department of Computer Science  
Aarhus University



## **Course evaluation**

*"The first lecture was intimidating  
and overwhelming"*

# Lecturer

Name	Gerth Stølting Brodal
Research	Algorithms and Data Structures (Computer Science)
Teaching	
2018 -	BSc course on Introduction to Programming with Scientific Applications
2003 -	BSc course on Introduction to Algorithms and Data Structures
1999 - 17	MSc courses on Computational Geometry, Algorithm Engineering, Advanced Data Structures, External Memory Algorithms and Data Structures
Python	Advanced Beginner

# Course description – [kursuskatalog.au.dk/en/course/130939/](https://kursuskatalog.au.dk/en/course/130939/)

## Introduction to Programming with Scientific Applications

### Description of qualifications

After the course the participants will have knowledge of principles and techniques for systematic **construction** of **programs**.

At the end of the course, the participants will be able to:

- apply constructions of a common programming language,
- develop **well-structured** programs and perform **testing** and **debugging** of these,
- explain fundamental programming concepts and basic algorithmic techniques,
- apply standard **tools for scientific applications**,
- use the documentation for a programming language and available software packages.

### Contents

The course gives an introduction to programming with scientific applications.

Programming concepts and techniques are introduced using the **Python** programming language.

The programming concepts are **illustrated in other programming languages**. The following content is included.

*Basic programming constructs:* Data types, operators, variables, flow of control, conditionals, loops, functions, recursion, scope, exceptions. *Object orientation:* Abstract data types, classes, inheritance, encapsulation. *Basic algorithmic techniques:* Sorting, binary search, dynamic programming. *Systematic development of programs:* Testing and debugging. File-based input/output, numerical analysis, functional programming. Scientific computing using standard packages for Python.

ECTS 10

### Hours - weeks - periods

Lectures 2 x 2 hours/week

TA sessions 1 x 3 hours/week

Study café 3 x 1 hour/week

### Language of instruction

Danish

### Instructor

Gerth Stølting Brodal

### Academic prerequisites

(Some) Linear algebra

### Exam

#### **5 hour programming**

Aid: Computer and Internet, headphones, no AI

7-point grading scale

### Prerequisites for examination participation

Submission and approval of 10 mandatory assignments and submission of

#### **1 implementation project**

**Notes** Grade reflects an overall assessment of implementation project and written examination. Project counts 20% and written exam counts 80%

# Question – Primary Education?

- a) Mathematics
- b) Mathematics-Economics
- c) Data Science
- d) Chemistry
- e) Physics
- f) Other Science-Technology
- g) Other

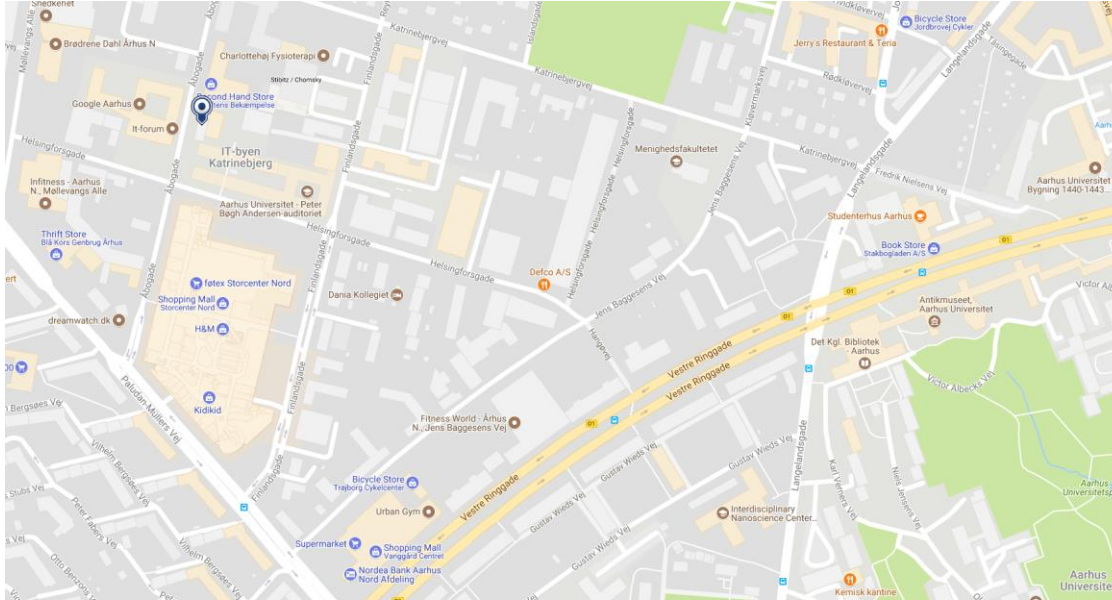
# Question – Programming languages you know?

# Question – Programming experience?

For the programming language you know best (if any) please state your proficiency level within the language.

- a) None
- b) Fundamental awareness (basic knowledge)
- c) Novice (limited experience)
- d) Intermediate (practical application)
- e) Advanced (applied theory)
- f) Expert (recognized authority)

Primary lecture material = slides




Week	Monday	Tuesday	Wednesday	Thursday	Friday
5	F1	no TA class	F2		
6	F3	TØ1	TØ1 / F4	TØ1	TØ1
7	F5	TØ2	TØ2 / F6	TØ2	TØ2
8	F7	TØ3	TØ3 / F8	TØ3	TØ3
9	F9	TØ4	TØ4 / F10	TØ4	TØ4
10	F11	TØ5	TØ5 / F12	TØ5	TØ5
11	F13	TØ6	TØ6 / F14	TØ6	TØ6
12	F15	TØ7	TØ7 / F16	TØ7	TØ7
13	F17	TØ8	TØ8 / F18	TØ8	TØ8
14	F19	TØ9	TØ9 / F20	TØ9	TØ9
15	F21	TØ10	TØ10 / F22	TØ10	TØ10
16	Easter break				
17		-	-	-	Kapsejlads?
18	F23	TØ11	TØ11 / F24	TØ11	TØ11
19	F25	TØ12	TØ12 / F26	TØ12	TØ12
20	F27	TØ13	TØ13 / -	TØ13	TØ13






	Monday	Tuesday	Wednesday	Thursday	Friday
8:15-9:00	TA meeting			MA1 (1Y)	
9:15-10:00	Study cafe		Study cafe		
10:15-11:00	Lecture		Lecture		MA2 (1Y)
11:15-12:00					
12:15-13:00				DV	Hold 2
13:15-14:00					
14:15-15:00		MA3 (2Y)	MØ1	Study cafe	
15:15-16:00		FY	MØ2		
16:15-17:00		Hold 1			
17:15-18:00					





# Course page on Brightspace and GitHub

 Introduktion til programmering med videnskabelige an...



Gerth Støtting Brodal  
as Student

Course Home Content Course Tools ▾ Classlist Zoom Panopto Help

gsbrodal.github.io/ipsa

Course information

Who, where and when


Handin deadlines

Course content on GitHub

## Course information

Welcome to the course *Introduction to Programming with Scientific Applications*

The information about the course is available on the Brightspace page and a public GitHub repository. You can use the discussion forum for questions, including the course content on GitHub.



## Introduction to Programming with Scientific Applications

Aarhus University, Department of Computer Science

### Welcome

Course plan

- Compact
- All slides (pdf)

Exercises

Handins

Final project

- I - Medians
- II - Portfolio
- III - NMR
- IV - MNIST
- V - Open topic

Exam

- Statistics
- Past exams

Plagiarism

Workload

Literature

Python installation

- Windows 11
- Mac & Linux

Python resources

- Books
- Videos

AU course description

## Welcome

This page contains the public content for the course *Introduction to Programming with Scientific Applications* offered by Aarhus University, Department of Computer Science. The lecturer is [Gerth Støtting Brodal](#).

Class lists, discussions, student handins and feedback are handled using Aarhus University's learning management system Brightspace at [brightspace.au.dk](#).

The course gives an introduction to the Python 3 programming language and applications using Python. Throughout the course students are encouraged to seek online information in e.g. the Python language specification.


The course will be run with weekly 2 x 2 hours lectures (alternatively recorded lectures on YouTube), 3 hours of exercise classes with a teaching assistant ("øvelser"), and 3 hours of staffed study café.

During the course students are required to hand in 10 weekly handins and one larger implementation project. Handins and the project are done in groups of up to three persons. Approval of the weekly handins is a prerequisite to attend the exam. The final exam will be a programming exam with all aids, incl. internet, and *the final grade will be based on an overall evaluation of the implementation project (20%) and the programming exam (80%)*.

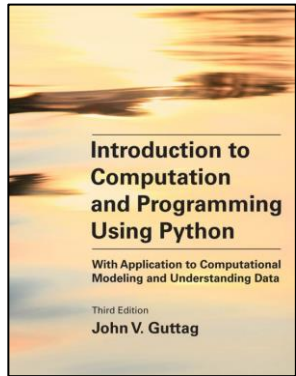
### Course content

The course gives an introduction to programming with scientific applications. Programming concepts and techniques are introduced using the Python programming language. The programming concepts are illustrated in other programming languages. The following content is included.

- Basic programming constructs: Data types, operators, variables, flow of control, conditionals, loops, functions, recursion, scope, exceptions.

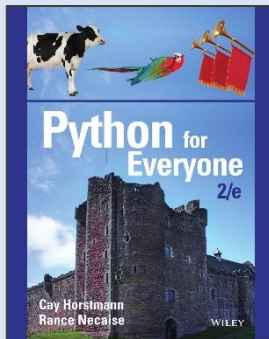


# Course text book – optional



John V. Guttag: **Introduction to Computation and Programming Using Python, Third Edition With Application to Computational Modeling and Understanding Data**. Third Edition. 664 pages. MIT Press, 2021.

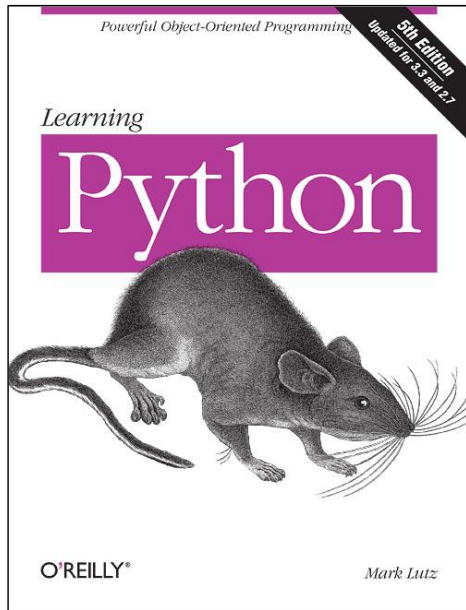
- *[Guttag, 2<sup>nd</sup> Ed., page 8] "The reader should be forewarned that this book is by no means a comprehensive introduction to Python". 3<sup>rd</sup> Ed. added about 80 pages on introduction to Python.*
- *Covers all basic features of Python enabling you to deal with data in Chapters 1-10 (212 pages) - remaining chapters are applications*
- *Other resources: Google, stackoverflow, Python.org, YouTube, Als...*



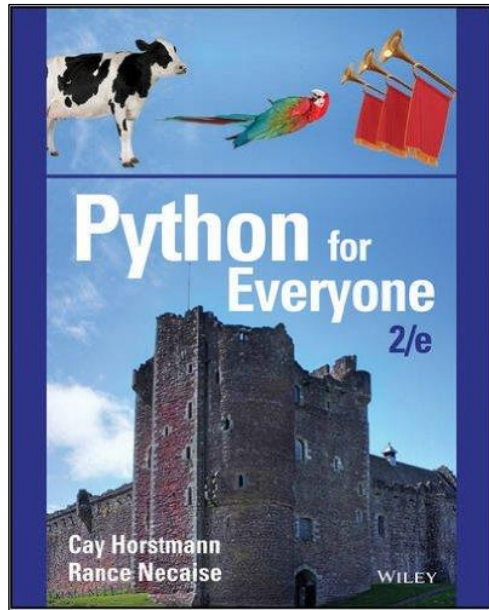
Comparison to a standard text book on the *programming language* Python by Cay Horstmann and Rance Necaise:

Topic **recursion** is covered by Guttag on page 123 (2<sup>nd</sup> edition on page 50), Horstmann and Necaise do it on page 611

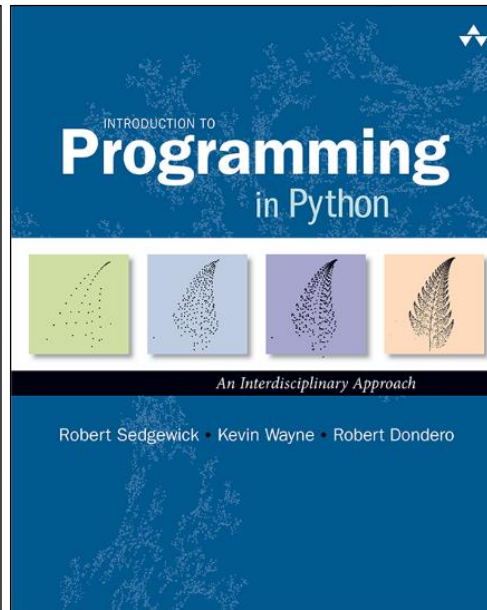
# Some other books on Python



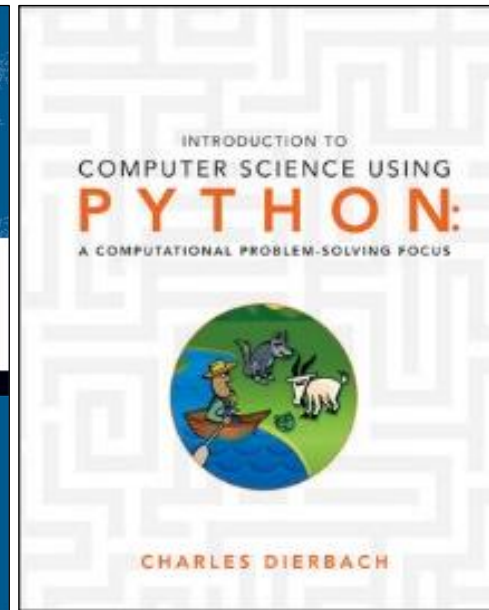
O'Reilly, 2013  
1684 pages



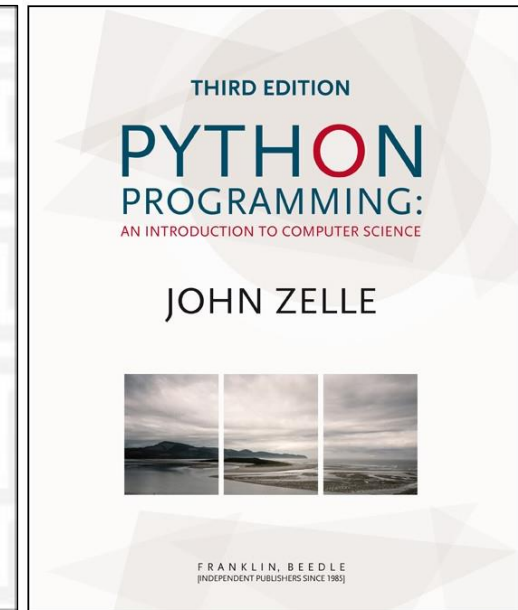
Wiley, 2016  
752 pages



Addison-Wesley, 2015  
794 pages



Wiley, 2013  
580 pages



Franklin & Beedle, 2016  
552 pages

... numerous online introduction texts/courses/videos on Python

Two Python programs

# A Python program

## Python shell

```
> x = 7
> print(x * x)
| 49
```

- 7 is an *integer literal* – in Python denoted an “int”
- x is the name of a *variable* that can hold some value
- = is assigning a value to a variable
- \* denotes multiplication
- print is the name of a built-in *function*,  
here we call print to print the result of 7 \* 7
- A program consists of a sequence of *statements*, executed sequentially

## Memory



# Question – What is the result of this program?

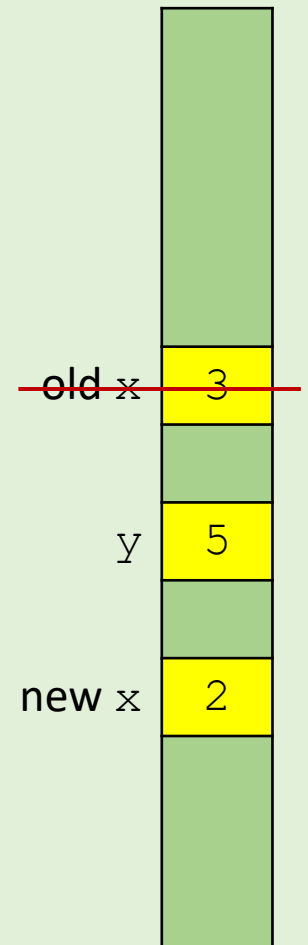
Python shell

```
> x = 3  
> y = 5  
> x = 2  
> print(x * y)
```

x assigned new value

- 😊 a) 10
- b) 15
- c) 25
- d) [15, 10]
- e) Error
- f) Don't know

Memory

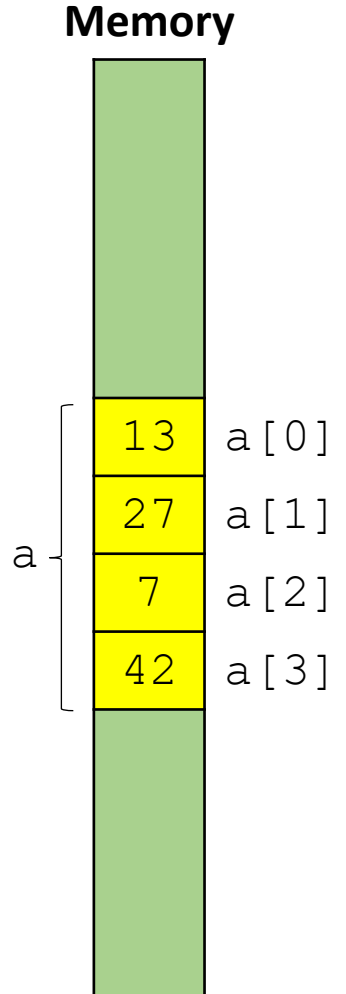


# Another Python program using lists

## Python shell

```
> a = [13, 27, 7, 42]
> print(a)
| [13, 27, 7, 42]
> print(a[2])
| 7
```

- `[13, 27, 7, 42]` is a *list* containing four integers
- `a[2]` refers to the entry in the list with *index* 2 (the first element has index 0, i.e. `a[2]` is the 3<sup>rd</sup> element of the list)
- Note that `print` also can print a list

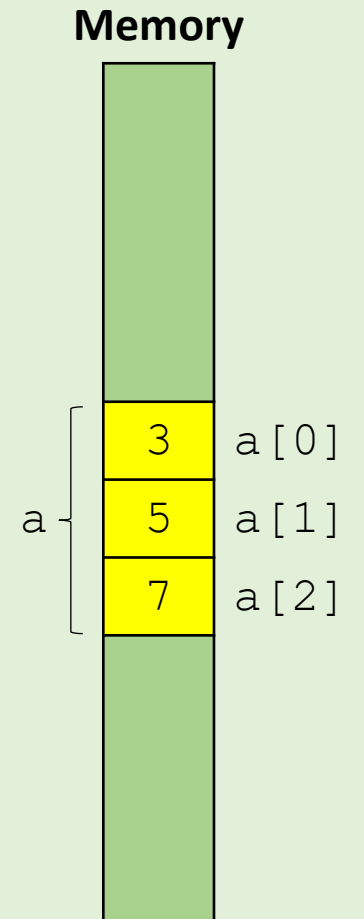


# Question – What is the result of this program?

Python shell

```
> a = [3, 5, 7]  
> print(a[1] + a[2])
```

- a) 8
- b) 10
- 😊 c) 12
- d) 15
- e) Don't know















# Why Python ?



the next slides will be technical

# TIOBE Index January 2025

Python #1  
Since  
November  
2021

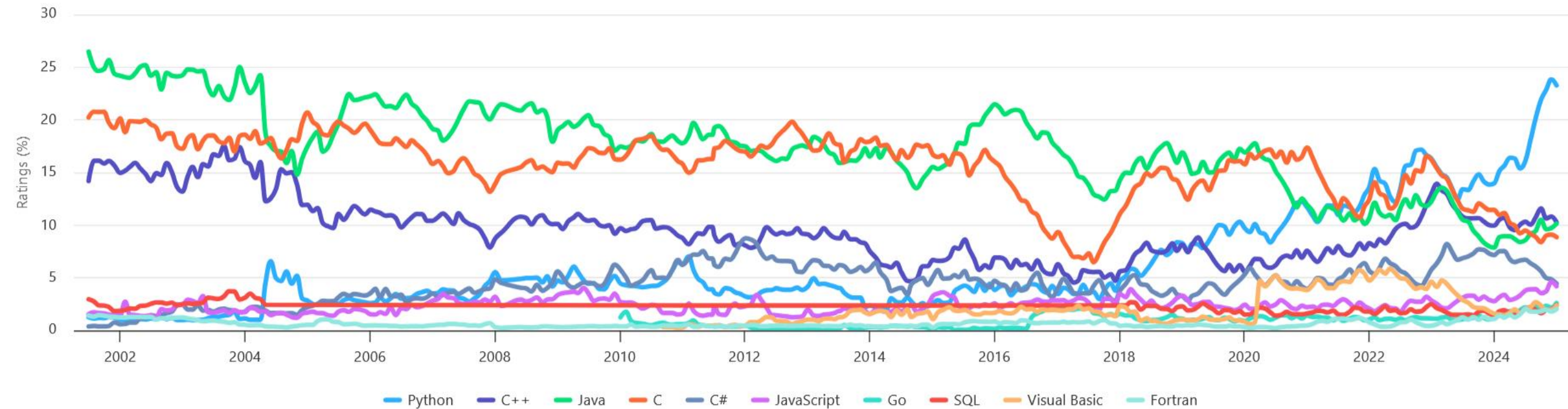
Jan 2025	Jan 2024	Change	Programming Language		Ratings	
1	1			Python	23.28%	+9.32%
2	3	^		C++	10.29%	+0.33%
3	4	^		Java	10.15%	+2.28%
4	2	v		C	8.86%	-2.59%
5	5			C#	4.45%	-2.71%
6	6			JavaScript	4.20%	+1.43%
7	11	^^		Go	2.61%	+1.24%
8	9	^		SQL	2.41%	+0.95%
9	8	v		Visual Basic	2.37%	+0.77%
10	12	^		Fortran	2.04%	+0.94%

The TIOBE Programming Community index is an indicator of the **popularity of programming languages**. The index is updated once a month. The ratings are based on the number of skilled engineers world-wide, courses and third party vendors. Popular search engines such as Google, Bing, Yahoo!, Wikipedia, Amazon, YouTube and Baidu are used to calculate the ratings. It is important to note that the TIOBE index is not about the *best* programming language or the language in which *most lines of code* have been written.

# Popularity of programming languages

## TIOBE Programming Community Index

Source: [www.tiobe.com](http://www.tiobe.com)



# “Hello World”

- In Java, C, C++ a lot of “{”, “}” and “;” are needed
- Java tends to have a lot of “public...” details that need to be spelled out
- Python is concise

## Java

```
public class HelloWorld {  
    public static void main( String[] args ) {  
        System.out.println( "Hello World!" );  
        System.exit( 0 );  
    }  
}
```

## C

```
#include <stdio.h>  
  
int main(int argc, char **argv) {  
    printf("Hello World");  
    return 0;  
}
```

## C++

```
#include <iostream>  
using namespace std;  
  
int main(int argc, char** argv) {  
    cout << "Hello, World!";  
    return 0;  
}
```

## Python 2

```
print "Hello world"
```

## Python 3

```
print("Hello world")
```

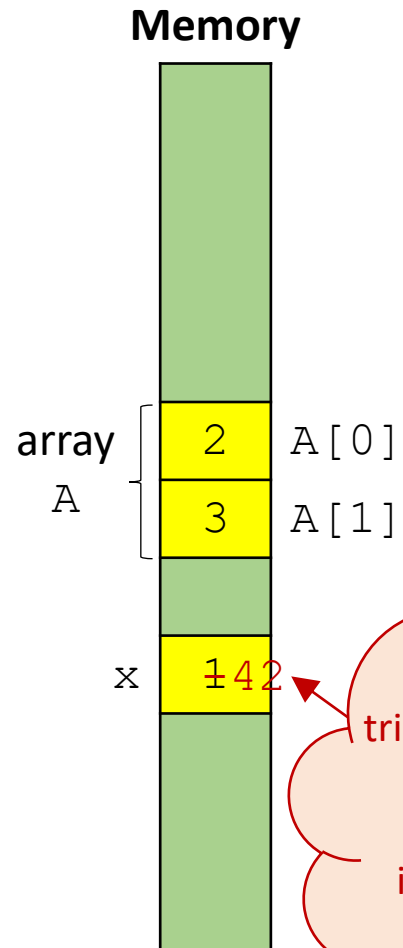
# Why Python ?

- **Short concise code**

# C index out of bounds

**Debugging** is the process of finding and resolving defects or problems within a computer program that prevent correct operation of computer software or a system.

*en.wikipedia.org/wiki/Debugging*



"A" only has size 2, but tries to update the 4<sup>th</sup> entry. No warning is giving. Something unexpected is overridden in memory. **Have fun debugging!**

## indexing.c

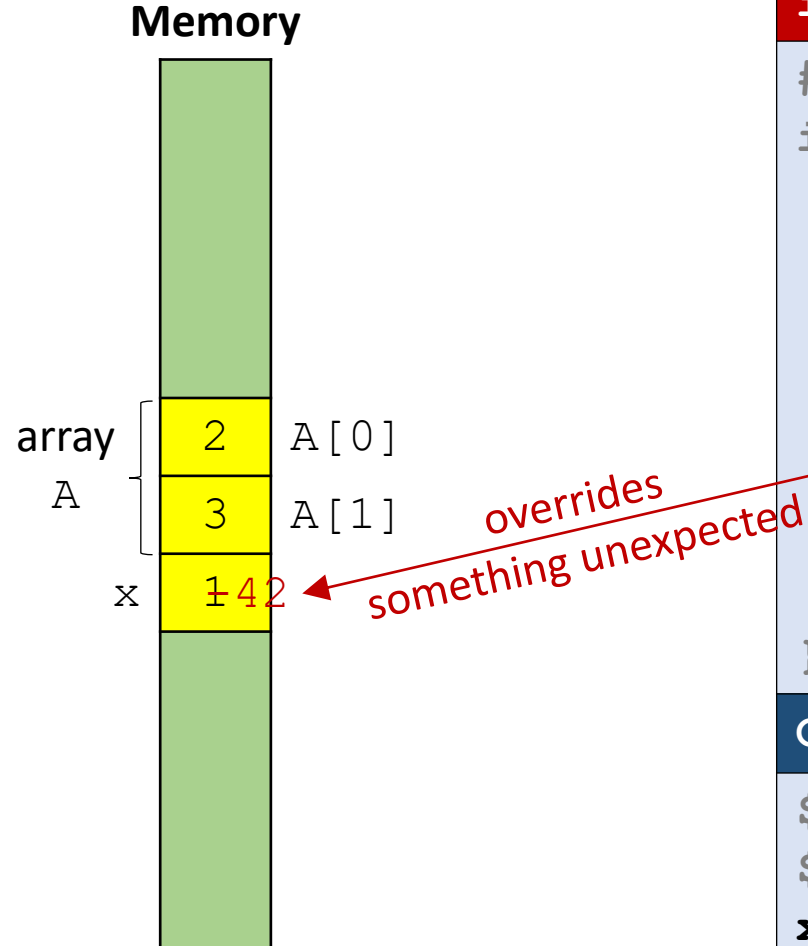
```
#include <stdio.h>
int main() {
    int x = 1;
    int A[2] = {2, 3}; // A[0] = 2, A[1] = 3
    printf("x = %d, A = {%d, %d}\n", x, A[0], A[1]);
    A[3] = 42; // index A[3] out of bounds
    printf("x = %d, A = {%d, %d}\n", x, A[0], A[1]);
    return 0;
}
```

## Output

```
$ gcc indexing.c
$ ./a.exe
x = 1, A = {2, 3}
x = 42, A = {2, 3}
```

Skipping checking for invalid indexing makes programs faster, but also requires disciplined programming

# ... and C++ index out of bounds



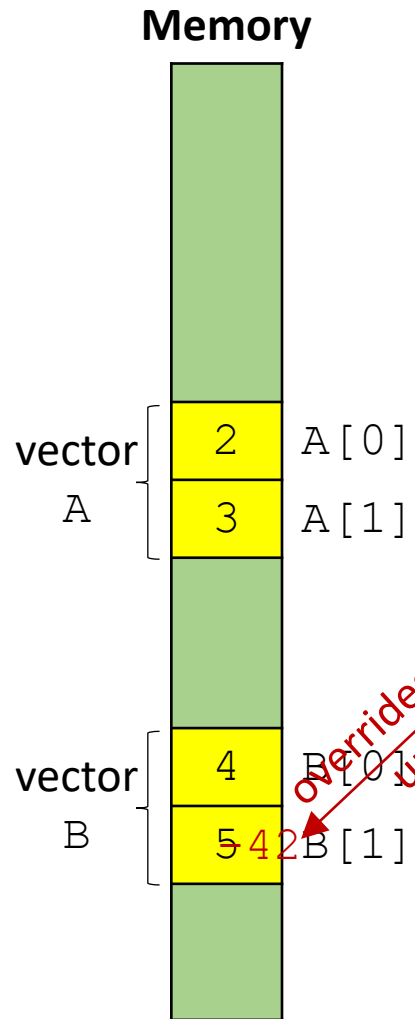
## indexing.cpp

```
#include <iostream>
int main() {
    int x = 1;
    int A[2] = {2, 3}; // A[0] = 2, A[1] = 3
    std::cout << "x = " << x << ", A = {"
                << A[0] << ", " << A[1] << "}" << std::endl;
    A[2] = 42; // index A[2] out of bounds
    std::cout << "x = " << x << ", A = {"
                << A[0] << ", " << A[1] << "}" << std::endl;
    return 0;
}
```

## Output

```
$ g++ indexing.cpp
$ ./a.exe
x = 1, A = {2, 3}
x = 42, A = {2, 3}
```

# ... and C++ `vector` index out of bounds



## indexing.cpp

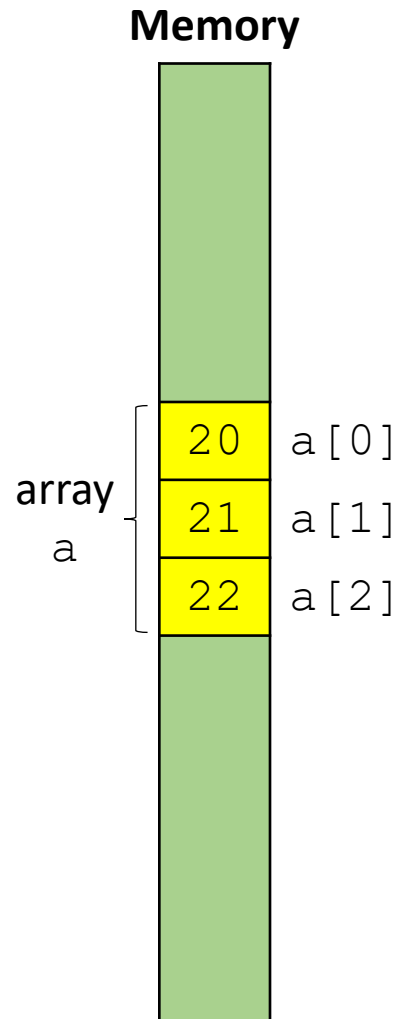
```
#include <iostream>
#include <vector>
int main() {
    std::vector<int> A = {2, 3}; // A[0] = 2, A[1] = 3
    std::vector<int> B = {4, 5}; // B[0] = 4, B[1] = 5
    std::cout << "A={" << A[0] << ", " << A[1] << "}, ";
    std::cout << "B={" << B[0] << ", " << B[1] << "}" << std::endl;
    A[9]=42; // index A[9] out of bounds
    std::cout << "A={" << A[0] << ", " << A[1] << "}, ";
    std::cout << "B={" << B[0] << ", " << B[1] << "}" << std::endl;
    return 0;
}
```

## Output

```
$ g++ -std=c++11 indexing-vector.cpp
$ ./a.exe
A={2, 3}, B={4, 5}
A={2, 3}, B={4, 42}
```



# ... and Java index out of bounds exception



## indexing.java

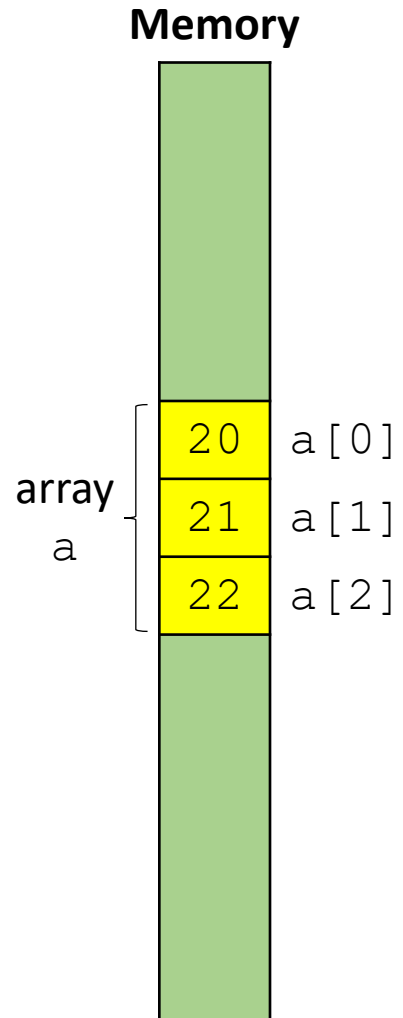
```
class IndexingTest{
    public static void main(String args[]){
        int a[] = {20, 21, 22};
        a[5] = 42; // index a[5] out of bounds
    }
}
```

## Output

```
$ javac indexing.java
$ java IndexingTest
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 5
    at IndexingTest.main(indexing.java:5)
```

Java provides error message when running the program

# ... and Python index out of bounds exception



## indexing.py

```
a = [20, 21, 22]
a[5] = 42 # index a[5] out of bounds
```

## Output


```
$ python indexing.py
Traceback (most recent call last):
  File "indexing.py", line 3, in <module>
    a[5] = 42
IndexError: list assignment index out of range
```

Python provides error message when running the program

# Memory safety

The White House 2024 | Press Release: “Future Software Should Be Memory Safe” ([www.whitehouse.gov](https://www.whitehouse.gov))

National Security Agency 2022 | Cybersecurity Information Sheet: Software Memory Safety ([media.defense.gov](https://media.defense.gov))

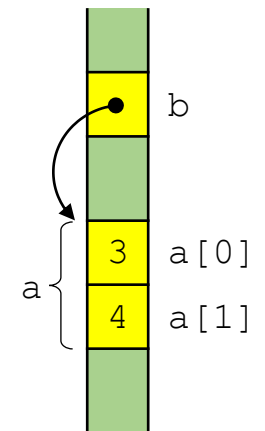
- C and C++ are flexible but **memory unsafe** programming languages
  - Unintended writes or reads to memory can be exploited by malicious cyber actors 
- Python, Java, Rust are examples of memory safe languages
- Rust aims at achieving the efficiency of C by slightly restricting flexibility

indexing.rs

```
fn main() {  
    let mut a = [3, 4];  
    a[2] = 7; // Compile error: this operation will panic at runtime  
    for i in 2..3 { a[i] = 7 } // Run-time panic: index out of bounds  
    let b = &mut a;  
    a[1] = 6; // Compile error: cannot use `a` because it was mutably borrowed  
    (*b)[0] = 5;  
    for i in 0..2 { println!("a[{}] = {}", i, a[i]) }  
}
```

[www.rust-lang.org](https://www.rust-lang.org) 

Memory



# Why Python ?

- Short concise code
- **Index out-of-range exceptions**

# C++ different ways to print a vector

vector-iterator.cpp

```
#include <iostream>
#include <vector>
int main() {
    // Vector is part of STL (Standard Template Library)
    std::vector<int> A = {20, 23, 26};
    // "C" indexing - since C++98
    for (int i = 0; i < A.size(); i++)
        std::cout << A[i] << std::endl;
    // iterator - since C++98
    for (std::vector<int>::iterator it = A.begin(); it != A.end(); ++it)
        std::cout << *it << std::endl;
    // "auto" iterator - since C++11
    for (auto it = A.begin(); it != A.end(); ++it)
        std::cout << *it << std::endl;
    // Range-based for-loop - since C++11
    for (auto e : A)
        std::cout << e << std::endl;
}
```

elegant

# Java - different ways to print a vector

vector-iterator.java

```
import java.util.Vector;
import java.util.Iterator;

class IteratorTest{
    public static void main(String[] args) {
        Vector<Integer> a = new Vector<Integer>();
        a.add(7);
        a.add(42);
        // "C" for-loop & get method
        for (int i = 0; i < a.size(); i++)
            System.out.println(a.get(i));
        // iterator
        for (Iterator it = a.iterator(); it.hasNext(); )
            System.out.println(it.next());
        // for-each loop - since Java 5
        for (Integer e : a)
            System.out.println(e);
    }
}
```

elegant

# The Python way to print a list

**print-list.py**

```
a = [20, 23, 26]
```

```
for e in a:  
    print(e)
```

**Output**

```
$ python print-list.py
```

```
20
```

```
23
```

```
26
```

# Why Python ?

- Short concise code
- Index out of range exceptions
- **Elegant for-each loop**



# C++ how not to print a vector

## print-vector.cpp

```
#include <iostream>
#include <vector>

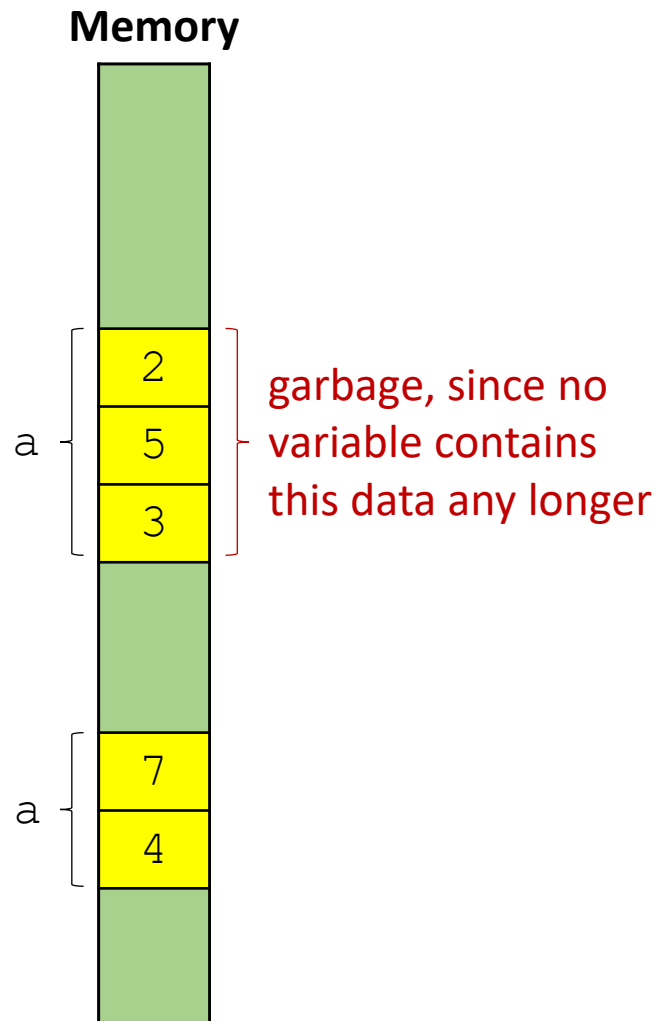
int main() {
    std::vector<int> A = {2, 3};
    std::cout << A << std::endl;
    return 0;
}
```

C++ vectors cannot be printed directly –  
mistake results in +200 lines of error messages

# Why Python ?

- Short concise code
- Index out of range exceptions
- Elegant for-each loop
- **Python hopefully better error messages than C++**

# Python and garbage collection



garbage.py

```
a = [2, 5, 3]
a = [7, 4]
```

a gets new value

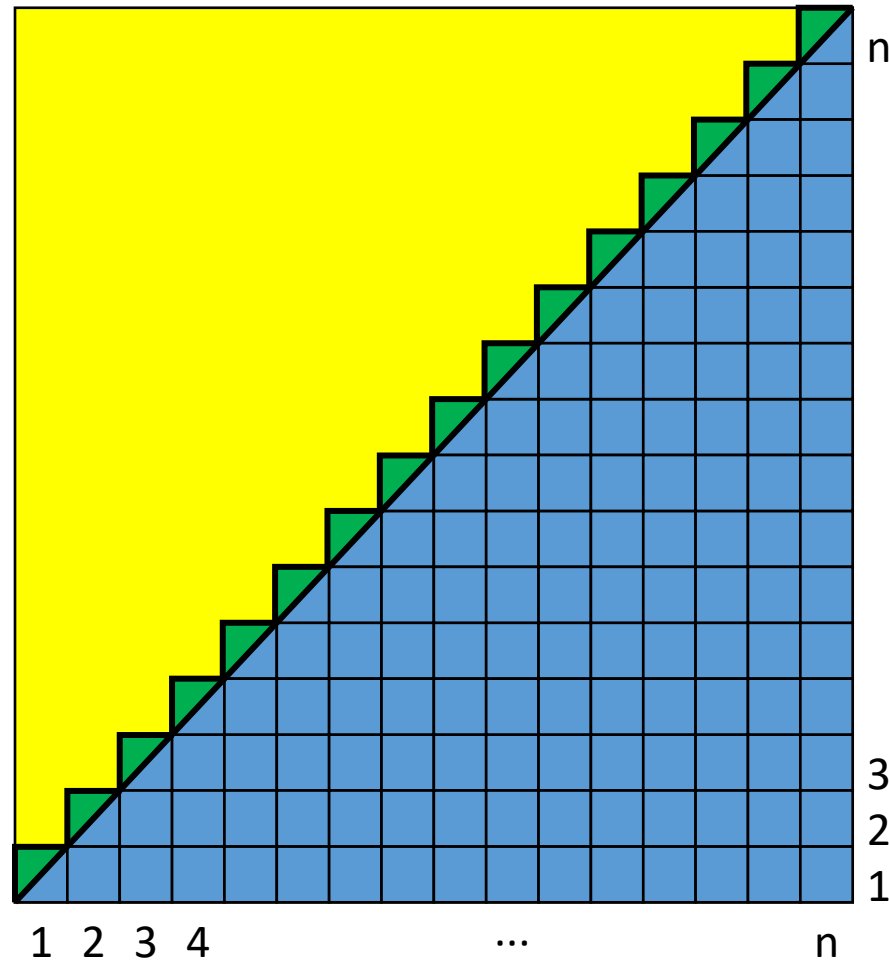
- Python and e.g. Java, C# and JavaScript have a **garbage collector** to automatically recycle garbage
- C and C++ garbage collection must be done explicitly by the program; forgetting to **free** memory again results in **memory leaks** – which can be really hard to find. **Have fun debugging!**
- Automatic garbage collection increases memory safety

# Why Python ?

- Short concise code
- Index out of range exceptions
- Elegant for-each loop
- Python hopefully better error messages than C++
- **Garbage collection is done automatically**

# Python performance vs C, C++ and Java

Compute sum  $1 + 2 + 3 + \dots + n = n^2/2 + n/2$



$$1 + 2 + \dots + n$$

#### add.py

```
import sys

n = int(sys.argv[1])
sum = 0
for i in range(1, n + 1):
    sum += i
print("Sum =", sum)
```

#### add.c

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    int n = atoi(argv[1]);
    int sum = 0;
    for (int i = 1; i <= n; i++)
        sum += i;
    printf("Sum = %d\n", sum);
}
```

#### add.cpp

```
#include <iostream>
#include <cstdlib>
using namespace std;

int main(int argc, char *argv[]) {
    int n = atoi(argv[1]);
    int sum = 0;
    for (int i = 1; i <= n; i++)
        sum += i;
    cout << "Sum = " << sum << endl;
}
```

#### add.java

```
class Add{
    public static void main(String args[]){
        int n = Integer.parseInt(args[0]);
        int sum = 0;
        for (int i = 1; i <= n; i++)
            sum += i;
        System.out.println("Sum = " + sum);
    }
}
```

# Timing results

Python							
n	C (gcc 9.2)	C++, int (g++ 9.2 )	C++, long (g++ 9.2 )	Java (12.0)	CPython (3.8.1)	PyPy (7.3.0)	Numba, int64
10 <sup>7</sup>	0.001 sec*	0.001 sec*	0.003 sec	0.006 sec*	1.5 sec	0.27 sec	0.002 sec
10 <sup>9</sup>	0.10 sec**	0.10 sec**	0.30 sec	0.40 sec**	145 sec	27 sec	0.2 sec

## Wrong output (overflow)

\* -2004260032 instead of 50000005000000

\*\* -243309312 instead of 500000000500000000

- since C, C++, and Java only uses 32 bits to represent integers (and 64 bits for "long" integers)

Have fun  
debugging!

```
Bit          6666666666555555555544444444443333333333222222222211111111110000000000
position     987654321098765432109876543210987654321098765432109876543210
bin(10**9)                                     111011100110101100101000000000
bin(50000005000000)                          1011010111100110001000100010010110101101000000
bin(-2004260032+2**32)                       10001000100010010110101101000000
bin(500000000500000000) 110111100000101101101011001111000101111110110010100000000
bin(-243309312+2**32)    11110001011111110110010100000000
```

# Timing results

n	C (gcc 9.2)	C++, int (g++ 9.2 )	C++, long (g++ 9.2 )	Java (12.0)	Python		
					Python (3.8.1)	PyPy (7.3.0)	Numba, int64
10 <sup>7</sup>	0.001 sec*	0.001 sec*	0.003 sec	0.006 sec*	1.5 sec	0.27 sec	0.002 sec
10 <sup>9</sup>	0.10 sec**	0.10 sec**	0.30 sec	0.40 sec**	145 sec	27 sec	0.2 sec

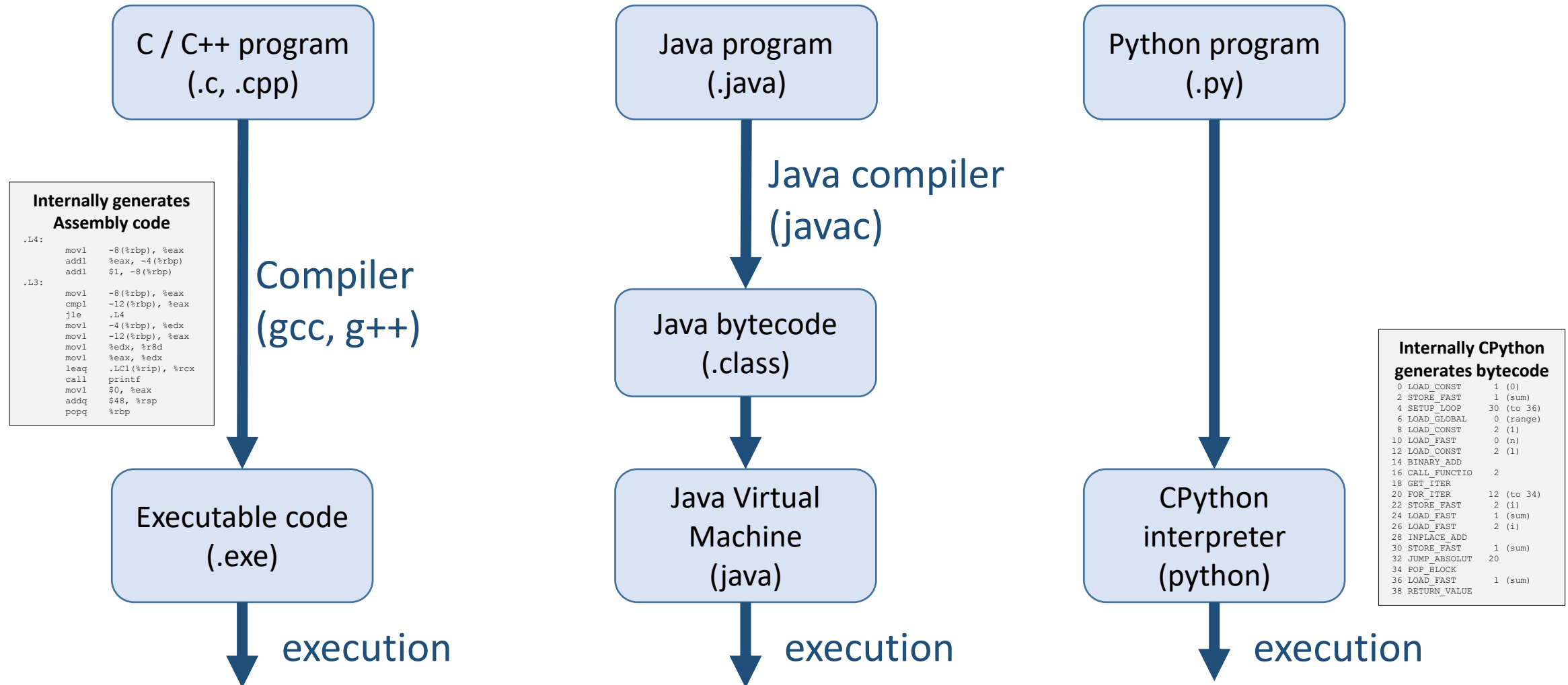
- Relative speed

**C ≈ C++ > Java >> Python**

- C, C++, Java need to care about integer overflows – select integer representation carefully with sufficient number of bits (8, 16, 32, 64, 128)
- Python natively works with arbitrary long integers (as memory on your machine allows). Also possible in Java using the class `java.math.BigInteger`
- Python programs can (sometimes) run faster using PyPy
- Number crunching in **Python** should be delegated to **specialized modules (e.g. Numpy, CPLEX, Numba)** – often written in C or C++ and requires selecting right integer representation

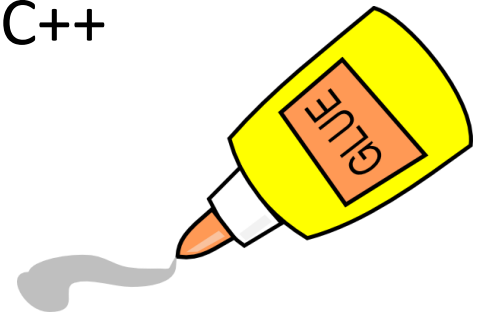


# Interpreter vs Compiler



# Why Python ?

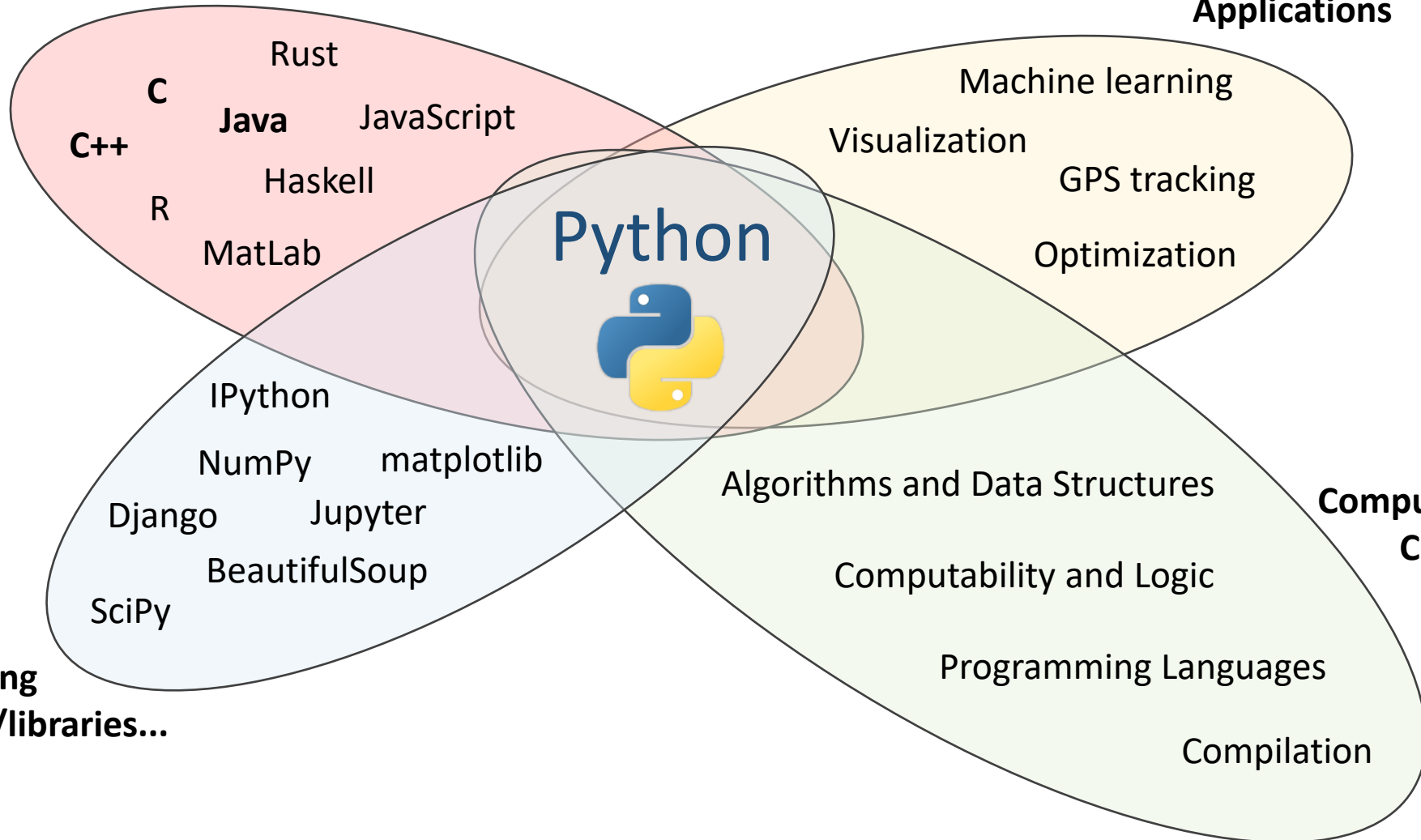
- Short concise code
- Index out of range exceptions
- Elegant for-each loop
- Python hopefully better error messages than C++
- Garbage collection is done automatically
- **Exact integer arithmetic (no overflows)**
- **Can delegate number crunching to C, C++, ...**



# This course

**Programming Languages**

**(Scientific)  
Applications**



**Programming  
modules/packages/libraries...**

# Course overview

Basic programming  
Advanced / specific python  
Libraries & applications

1. Introduction to Python	10. Functions as objects	19. Linear programming
2. Python basics / if	11. Object oriented programming	20. Generators, iterators, with
3. Basic operations	12. Class hierarchies	21. Modules and packages
4. Lists / while / for	13. Exceptions and files	22. Working with text
5. Tuples / comprehensions	14. Doc, testing, debugging	23. Relational data
6. Dictionaries and sets	15. Decorators	24. Clustering
7. Functions	16. Dynamic programming	25. Graphical user interfaces (GUI)
8. Recursion	17. Visualization and optimization	26. Java vs Python
9. Recursion and Iteration	18. Multi-dimensional data	27. Final lecture

10 handins  
1 final project (last 1 month)

# History of Python development

- Python created by Guido van Rossum in 1989, first release 0.9.0 1991
- Python 2 → Python 3 (clean up of Python 2 language)
  - Python 2 – version 2.0 released 2000, final version 2.7 released mid-2010
  - Python 3 – released 2008, current release 3.13.1
- Python 3 is *not* backward compatible, libraries incompatible

Python 2	Python 3
<code>print 42</code>	<code>print(42)</code>
<code>int</code> = C long (32 bits)	<code>int</code> = arbitrary number of digits (= named “long” in Python 2)
<code>7/3</code> → 2 returns “int”	<code>7/3</code> → 2.333... returns “float”
<code>range()</code> returns list (memory intensive)	<code>range()</code> returns iterator (memory efficient; xrange in Python 2)

# Python.org

The screenshot shows the Python.org homepage with a dark blue header and a lighter blue main content area. The header includes navigation links: Python, PSF, Docs, PyPI, Jobs, and Community. The main content area features the Python logo, a search bar, and a 'Donate' button. Below the header is a secondary navigation bar with links: About, Downloads, Documentation, Community, Success Stories, News, and Events. The main content area is divided into two columns. The left column displays a code snippet for a for loop on a list, with a red arrow pointing to it labeled 'Download Python and IDLE'. The right column features an article titled 'All the Flow You'd Expect' with a red arrow pointing to it labeled '+500.000 Python packages'. Below the article is a pagination bar with numbers 1 through 5. At the bottom of the page is a footer with four columns: 'Get Started', 'Download', 'Docs', and 'Jobs'. Each column contains a brief description and a link to the relevant resource. Red arrows from the main content area point to the 'Docs' and 'PyPI' links in the header.

Python.org

Python

PSF

Docs

PyPI

Jobs

Community

python™

Donate

GO

Socialize

About

Downloads

Documentation

Community

Success Stories

News

Events

```
# For loop on a list
>>> numbers = [2, 4, 6, 8]
>>> product = 1
>>> for number in numbers:
...     product = product * number
...
>>> print('The product is:', product)
The product is: 384
```

>\_

### All the Flow You'd Expect

Python knows the usual control flow statements that other languages speak — `if`, `for`, `while` and `range` — with some of its own twists, of course. [More control flow tools in Python 3](#)

1 2 3 4 5

Python is a programming language that lets you work quickly and integrate systems more effectively. >>> [Learn More](#)

#### Get Started

Whether you're new to programming or an experienced developer, it's easy to learn and use Python.

[Start with our Beginner's Guide](#)

#### Download

Python source code and installers are available for download for all versions!

Latest: [Python 3.11.1](#)

#### Docs

Documentation for Python's standard library, along with tutorials and guides, are available online.

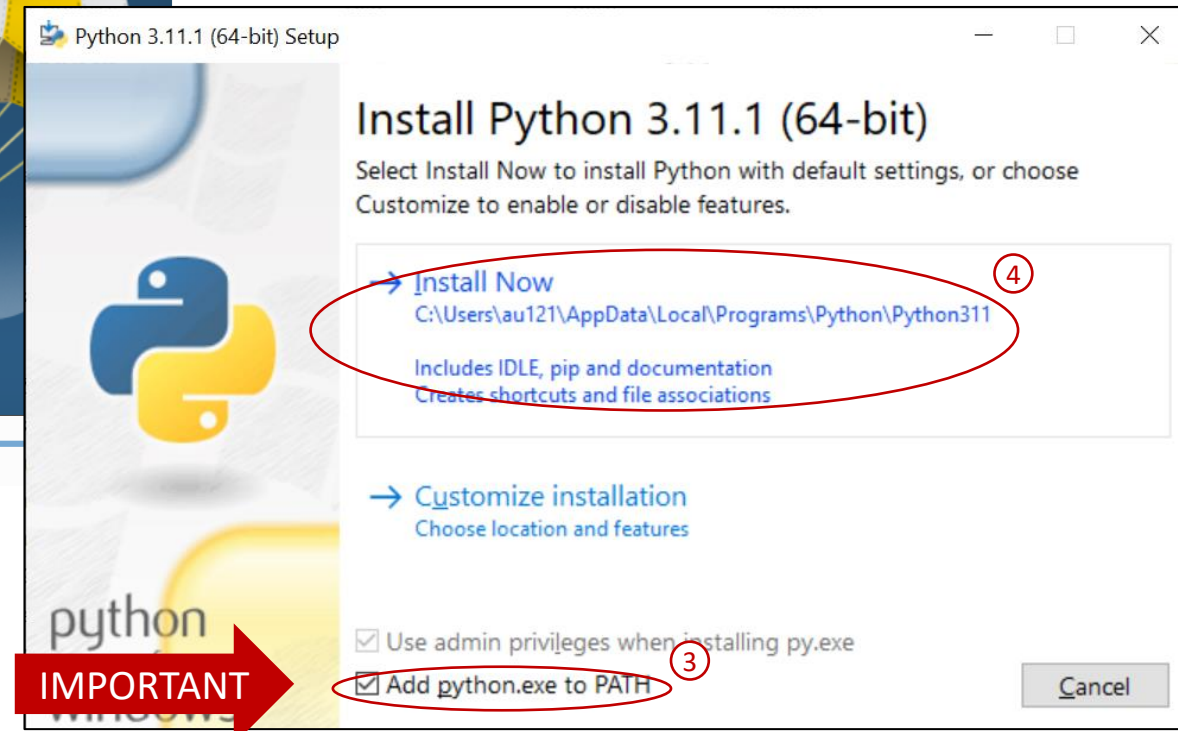
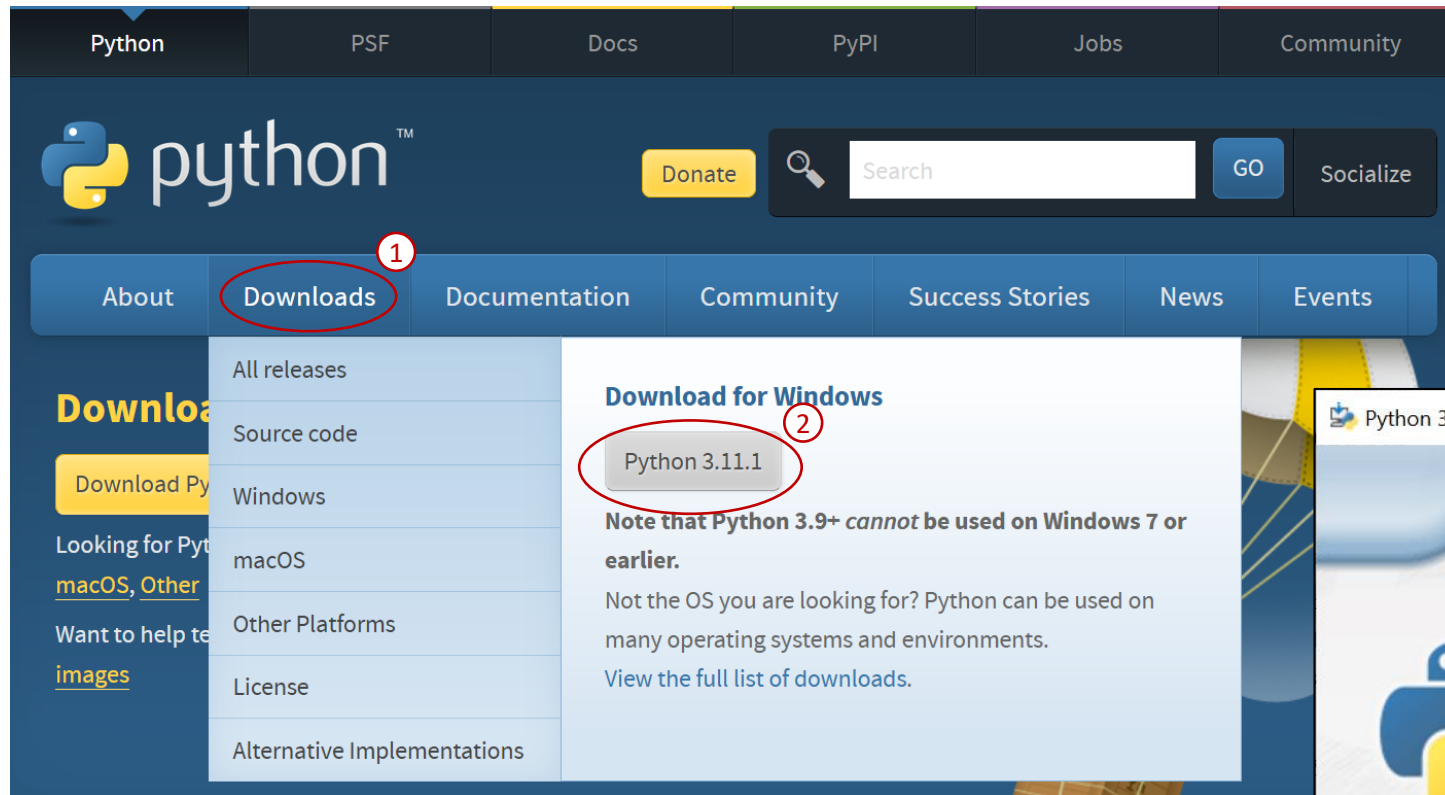
[docs.python.org](#)

#### Jobs

Looking for work or have a Python related position that you're trying to hire for? Our **relaunched community-run job board** is the place to go.

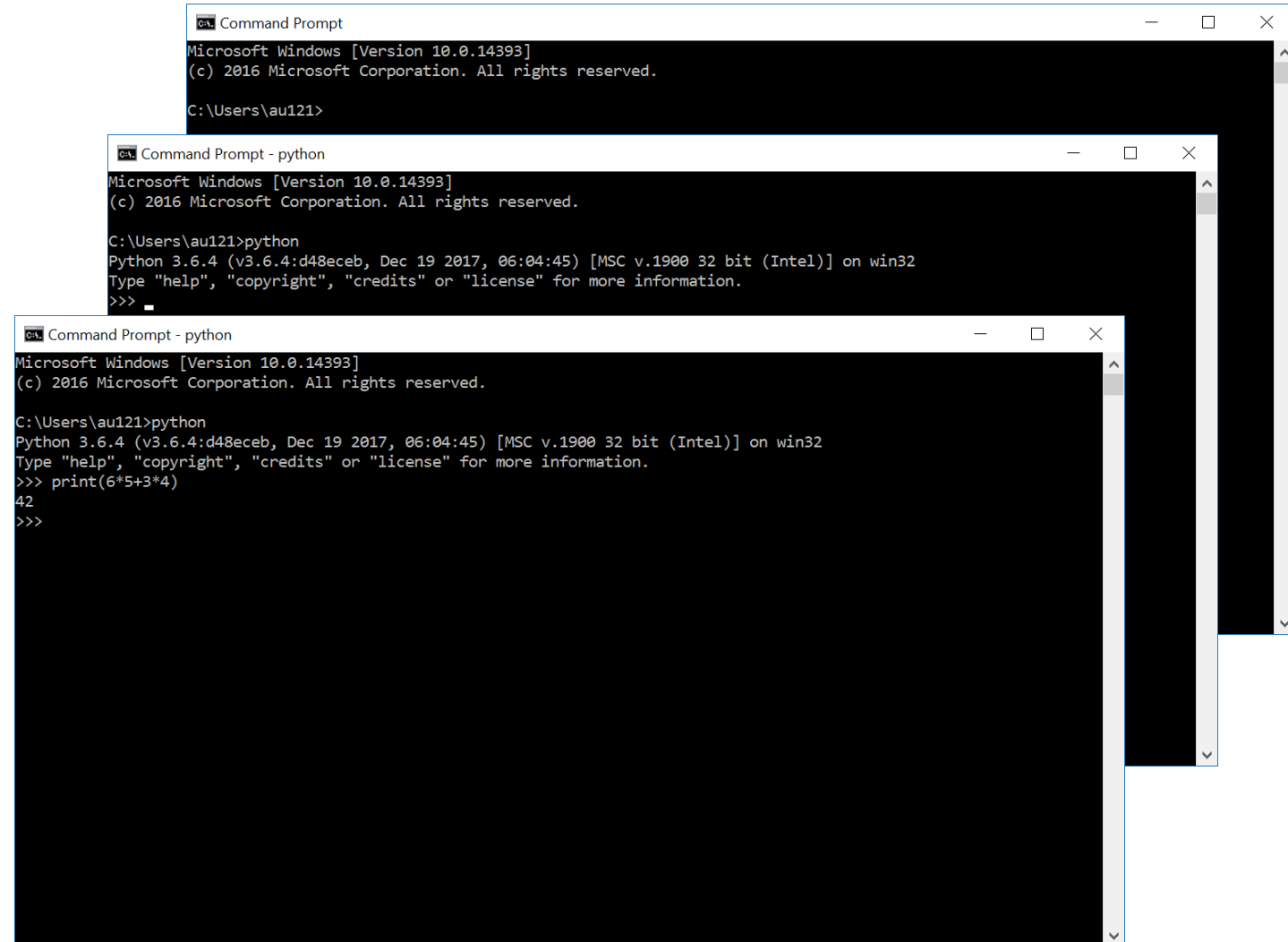
[jobs.python.org](#)

# Installing Python



# Running the Python Interpreter from a terminal

- Open Command Prompt (Windows-key + cmd)
- Type “python” + return
- Start executing Python statements
- To exit shell:  
Ctrl-Z + return *or*  
exit() + return
- Note: Sometimes “python” is installed as “python3”



The image displays three overlapping screenshots of a Windows Command Prompt window, illustrating the process of running the Python interpreter. The top window shows the standard Windows boot-up text and the user's prompt. The middle window shows the user typing 'python' and the system outputting the Python version and build information. The bottom window shows the user typing a print statement and the system outputting the result.

```
Command Prompt
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\au121>

Command Prompt - python
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\au121>python
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>

Command Prompt - python
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\au121>python
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print(6*5+3*4)
42
>>>
```



# Installing IPython –

## A more powerful interactive Python shell

- Open Command Prompt

- Execute:

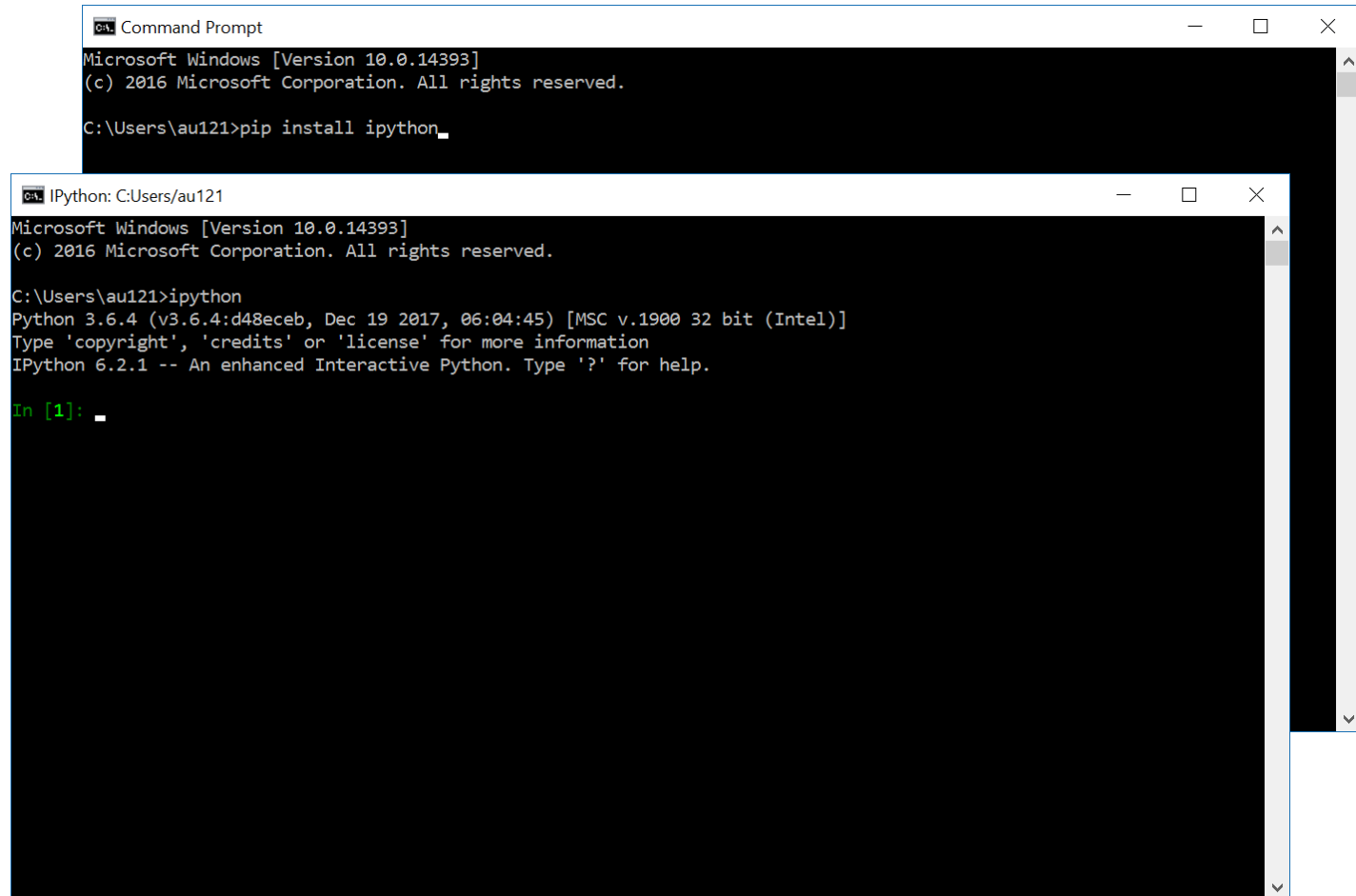
```
pip install ipython
```

- Start IPython

```
ipython
```

- pip = the Python package manager

- Note: Sometimes “pip” is installed as “pip3”



The screenshot displays two overlapping windows. The top window is titled 'Command Prompt' and shows the command `pip install ipython` being entered at the `C:\Users\au121>` prompt. The bottom window is titled 'IPython: C:\Users\au121' and shows the output of the installation, including the Python version (3.6.4) and IPython version (6.2.1). The IPython prompt `In [1]:` is visible at the bottom of the window.

```
Command Prompt
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\au121>pip install ipython_

IPython: C:\Users\au121
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\au121>ipython
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
Type 'copyright', 'credits' or 'license' for more information
IPython 6.2.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]:
```

# Some other usefull packages

- Try installing some more Python packages:

```
pip install numpy
```

linear algebra support (N-dimensional arrays)

```
pip install scipy
```

numerical integration and optimization

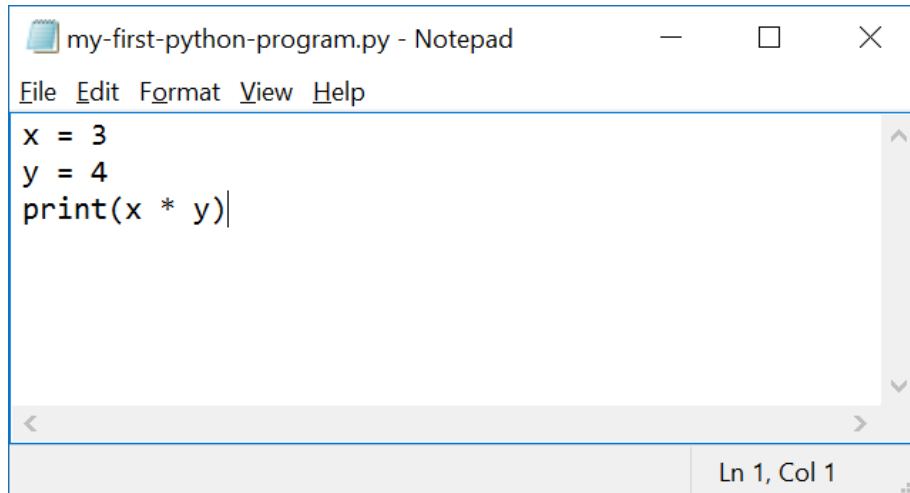
```
pip install matplotlib
```

2D plotting library

```
pip install pylint
```

Python source code analyzer enforcing a coding standard

# Creating a Python program the very basic way

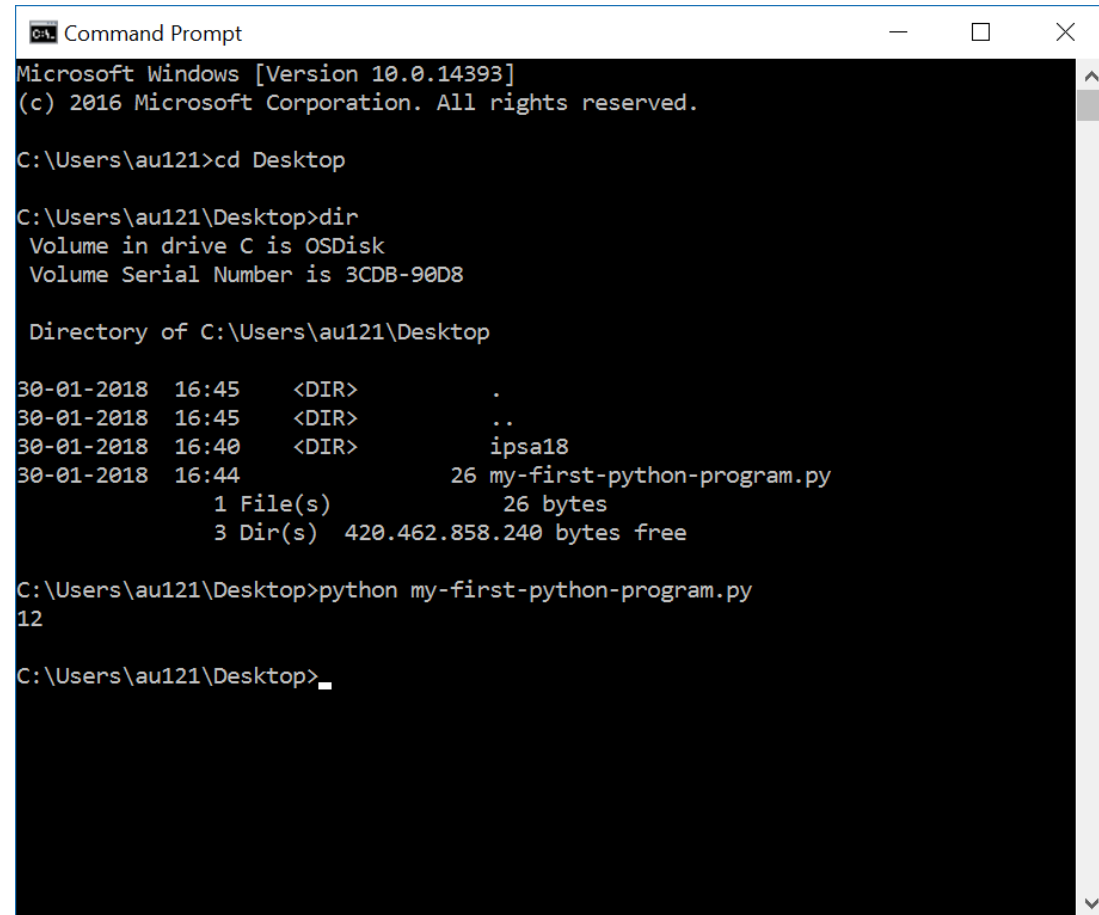
A screenshot of a Notepad window titled "my-first-python-program.py - Notepad". The window contains the following Python code:

```
x = 3
y = 4
print(x * y)
```

The status bar at the bottom right indicates "Ln 1, Col 1".

```
my-first-python-program.py - Notepad
File Edit Format View Help
x = 3
y = 4
print(x * y)
Ln 1, Col 1
```

- Open Notepad (or TextEdit on Mac)
  - write a simple Python program
  - save it
- Open a command prompt
  - go to folder (using cd)
  - run the program using  
`python <program name>.py`

A screenshot of a Windows Command Prompt window. The user navigates to the Desktop directory and lists its contents, showing the file "my-first-python-program.py". Then, the user runs the command "python my-first-python-program.py", which outputs the number "12".

```
Command Prompt
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\au121>cd Desktop

C:\Users\au121\Desktop>dir
Volume in drive C is OSDisk
Volume Serial Number is 3CDB-90D8

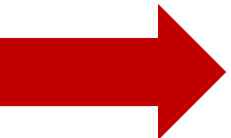
Directory of C:\Users\au121\Desktop

30-01-2018  16:45    <DIR>          .
30-01-2018  16:45    <DIR>          ..
30-01-2018  16:40    <DIR>          ipsa18
30-01-2018  16:44                26 my-first-python-program.py
               1 File(s)                26 bytes
               3 Dir(s)  420.462.858.240 bytes free

C:\Users\au121\Desktop>python my-first-python-program.py
12

C:\Users\au121\Desktop>
```

# ... or open IDLE and run program with F5



enable  
line numbers  
under options

```
my-first-python-program.py - C:\Users\au121\Desktop\my-first-python-program.py (3.11.0)
File Edit Format Run Options Window Help
1 x = 3
2 y = 4
3 print(x * y)
4 |
Ln: 4 Col: 0
```

```
IDLE Shell 3.11.0
File Edit Shell Debug Options Window Help
Python 3.11.0 (main, Oct 24 2022, 18:26:48) [MSC v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\au121\Desktop\my-first-python-program.py =====
12
>>>
Ln: 6 Col: 0
```

- IDLE ships with Python from python.org
- Good beginner IDE (Integrated Development Environment)

# The Python Ecosystem

- **Interpreters/compiler**

- CPython – reference C implementation from python.org
- PyPy – written in RPython (a subset of Python) – faster than Cpython
- Jython – written in Java and compiles to Java bytecode, runs on the JVM
- IronPython – written in C#, compiles to Microsoft's Common Language Runtime (CLR) bytecode
- Cython – project translating Python-ish code to C

- **Shells (IPython, IDLE, Jupyter)**

- **Libraries/modules/packages**

- [pypi.python.org/pypi](https://pypi.python.org/pypi) (PyPI - the Python Package Index, +500.000 packages)

- **IDEs (Integrated development environment)**

- IDLE comes with Python ([docs.python.org/3/library/idle.html](https://docs.python.org/3/library/idle.html))
- Anaconda w. Spyder, IPython ([www.anaconda.com/download](https://www.anaconda.com/download))
- Canopy ([enthought.com/product/canopy](https://enthought.com/product/canopy))
- Visual Studio Code ([code.visualstudio.com](https://code.visualstudio.com))
- Python tools for Visual Studio ([github.com/Microsoft/PTVS](https://github.com/Microsoft/PTVS))
- PyCharm ([www.jetbrains.com/pycharm/](https://www.jetbrains.com/pycharm/))
- Emacs (Python mode and ElPy mode)
- Notepad++

← Good beginner Python IDE

← *“Visual Studio Code is used by more than twice as many developers than its nearest alternative”, [Stack overflow survey 2024](#)*

Try to google “[best ide python](#)”

- **Python Style guide (PEP8)**


- pylint, pep8, flake8

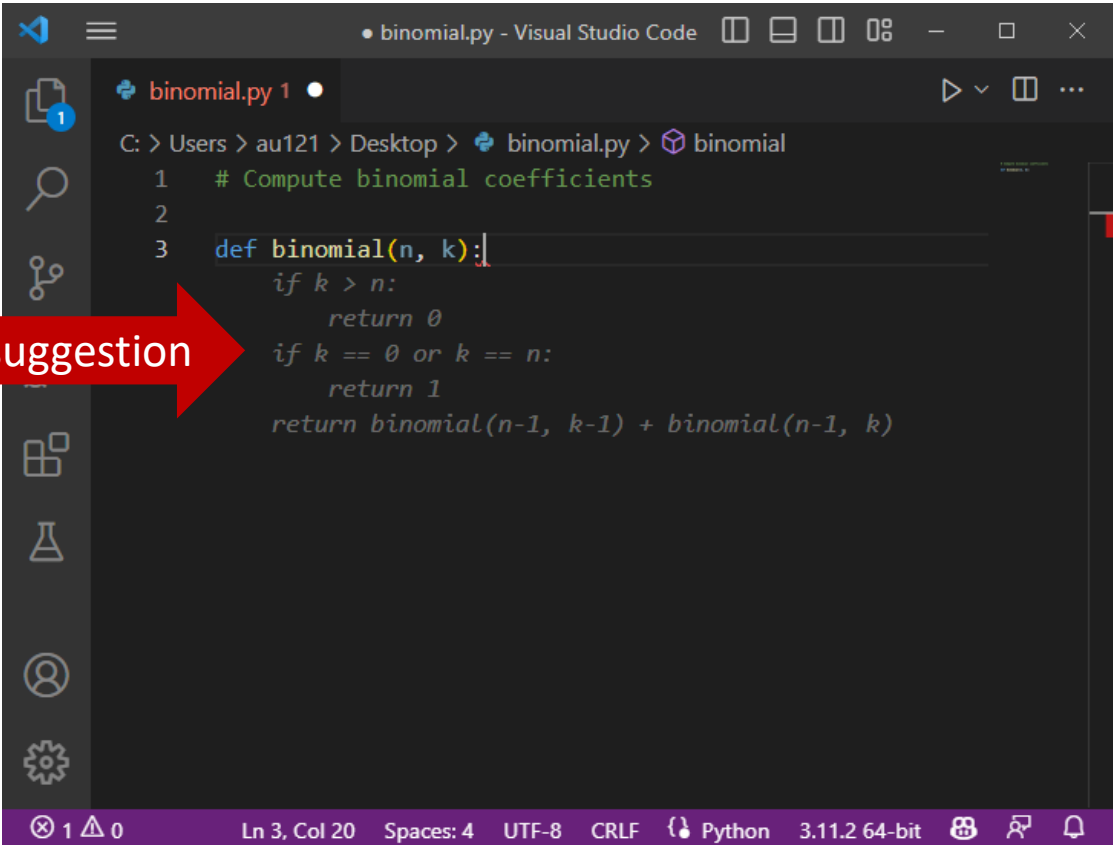
- **Python online**

- Google colab ([colab.research.google.com](https://colab.research.google.com)), repl.it, sagemath.org, ...



# IDEs and AI assistants

- Some IDEs integrate AI assistants to support code suggestions, e.g. GitHub Copilot in VS Code
- AI assistants increase productivity *if* you understand their output
-  ■ AI assistants are not allowed at the exam



```
binomial.py 1
C: > Users > au121 > Desktop > binomial.py > binomial
1 # Compute binomial coefficients
2
3 def binomial(n, k):
    if k > n:
        return 0
    if k == 0 or k == n:
        return 1
    return binomial(n-1, k-1) + binomial(n-1, k)
```

Guido van Rossum, inventor of Python, on GitHub Copilot

*"I use it every day. It writes a lot of code for me... and usually it is slightly wrong but it still saves me typing."*

Python and the Future of Programming, Guido van Rossum interviewed by Lex Fridman