Introduction to Computer Science for Geographers

Scientific Programming Languages

Christina Ludwig

Seminar im Sommersemester 2019

Arbeitsgruppe Geoinformatik,
Geographisches Institut

UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

Time table

Time	Day 1	Day 2	Day 3	Day 4	Day 5
9:15 – 9:45	Introduction	Questions	Questions	Questions	Questions
9:45 - 10:45		Computer Hardware	Program Design	Errors, Debugging, Testing	APIs
10:45 - 11:00	BREAK	BREAK	BREAK	BREAK	BREAK
11:00 - 12:30	**Programming Languages**	Numpy	Program Design	Errors, Debugging, Testing	Open Space
12:30 - 13:45	LUNCH	LUNCH	LUNCH	LUNCH	LUNCH
13:45 - 15:15	Vector data in Python	Raster data in Python	Documentation and code style	Profiling & Efficiency	Final Assignment
15:15- 15:30	BREAK	BREAK	BREAK	BREAK	BREAK
15:30 - 16:15	git session	git session	git collaboration	Profiling & Efficiency	
16:15 - 16:30	Questions + Feedback	Questions + Feedback	Questions + Feedback	Questions + Feedback	

Learning Goals

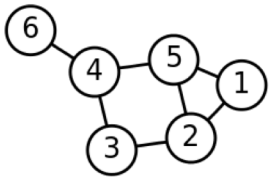
1. Explain the difference between compilation and interpretation of source code.
2. Name advantages and disadvantages of using C vs. Python.
3. Use the command line to navigate through the file system.
4. Set up a Python environment using Anaconda.

Agenda

1. Types of Programming Languages
2. Compilation vs. Interpretation
3. Comparison of C vs. Python
4. Virtual Python Environments using Anaconda

How to solve a problem using the computer

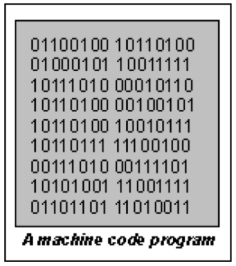
Problem | Instructions | Source code | Running instance | Output



1. Go to...
2. Do XY
3. ...

```
def add5(x):
    return x+5

def dotwrite(ast):
    nodename = getNodename()
    label=symbol.sym_name.get(int(ast[0]),ast[0])
    print '    %s [label="%s' % (nodename, label),
    if isinstance(ast[1], str):
        if ast[1].strip():
            print '= %s"];' % ast[1]
        else:
            print '"]'
    else:
        print '"];'
        children = []
        for  in n, childenumerate(ast[1:]):
            children.append(dotwrite(child))
        print ,'    %s -> {' % nodename
        for  in :namechildren
            print '%s' % name,
```



A machine code program

42

Problem → Algorithm → Program → Process → Output

What is happening here?

A **programming language** is a formal language, which comprises a set of instructions that produce various kinds of output. [Wikipedia]

Might also be a real-world process
e.g. Hydrological run-off model

Generations of Programming Languages (GL)

Natural Language (high-level)

Level of Abstraction

Computer Language (low-level)

Self learning/stochastic Language (5.GL)

Machine determines a solution itself (not widely used)

High-Level Language (4.GL)

e.g. R, SQL, MATLAB

High-Level Language (3.GL)

e.g. C, C++, C#, Java

```
func sum(n: Int): Int = {
    n * (n + 1) / 2
}
```

Source code

Assembly language (2.GL)

specific to a certain processor

```
LEA $eax, n
ADD $eax, 1
MUL $eax, n
DIV $eax, 2
RET $eax
```

Assembly code

Machine Language (1. GL)

specific to a certain processor

```
10100111100
11110011001
10010010010
10110111001
11101111011
```

Machine code

Programming for Scientists

Scientists (or Non-Computer Scientist) usually use High-Level programming languages.

The question is usually just about whether to use a

compiled or interpreted

language.

C, C#, C++

Python

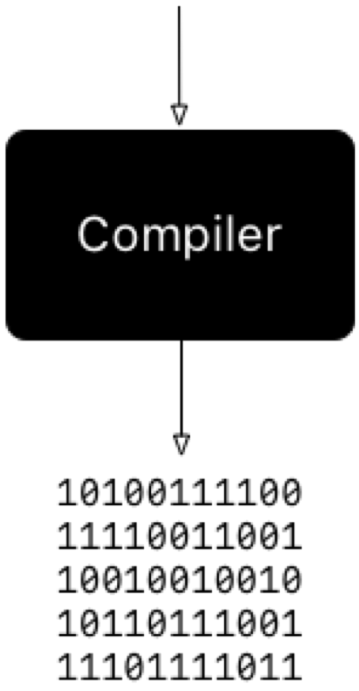
Do you know examples for compiled and interpreted languages?

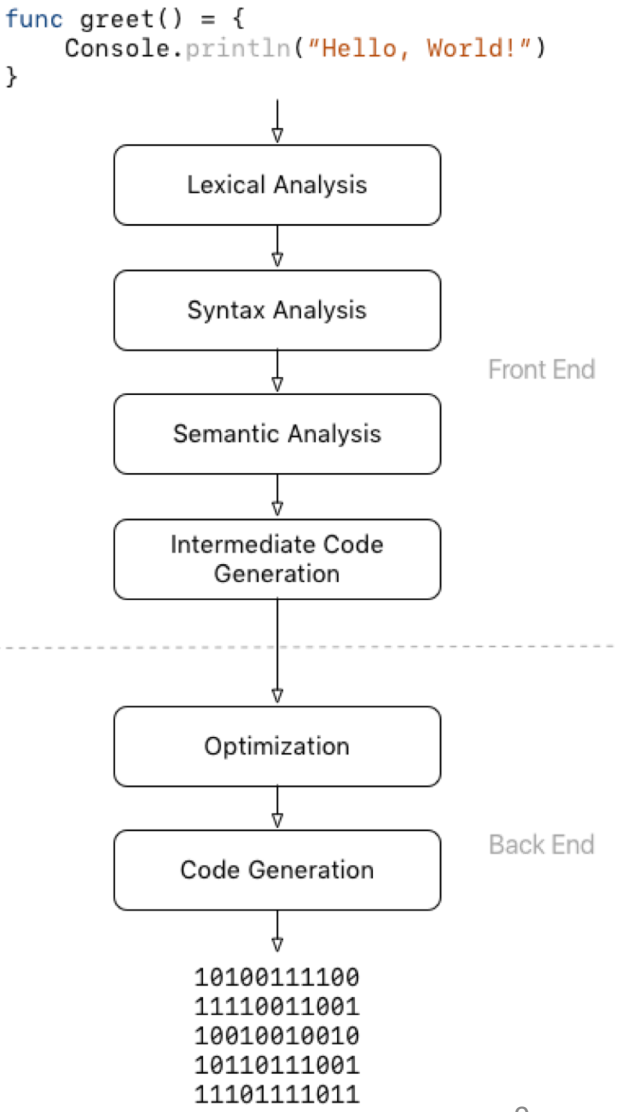
Agenda

1. Types of Programming Languages
2. Compilation vs. Interpretation
3. Comparison of C vs. Python
4. Virtual Python Environments using Anaconda

What is Compilation?

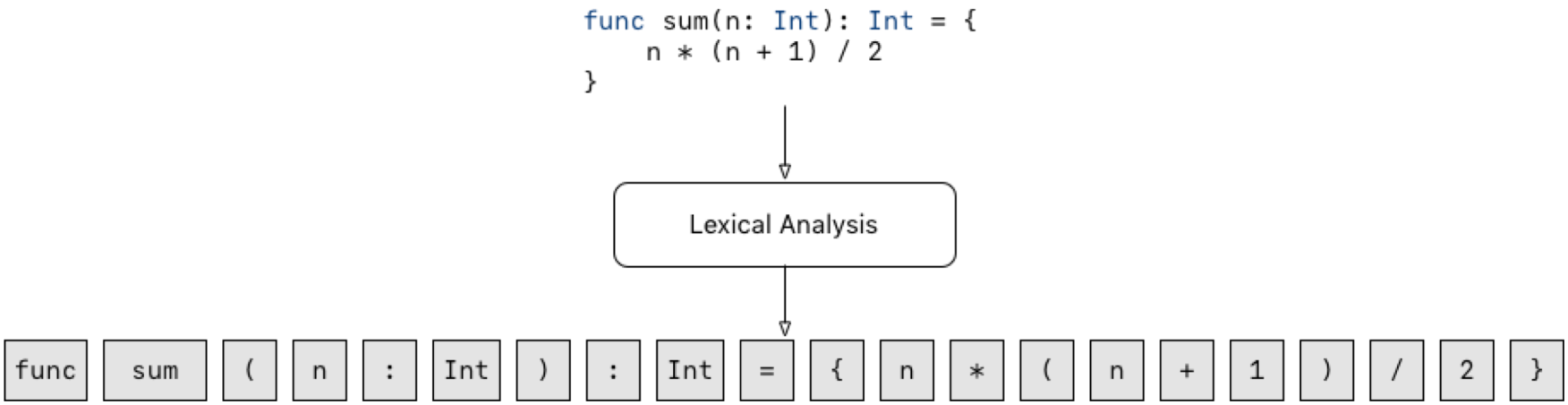
```
func greet() = {
    Console.println("Hello, World!")
}
```





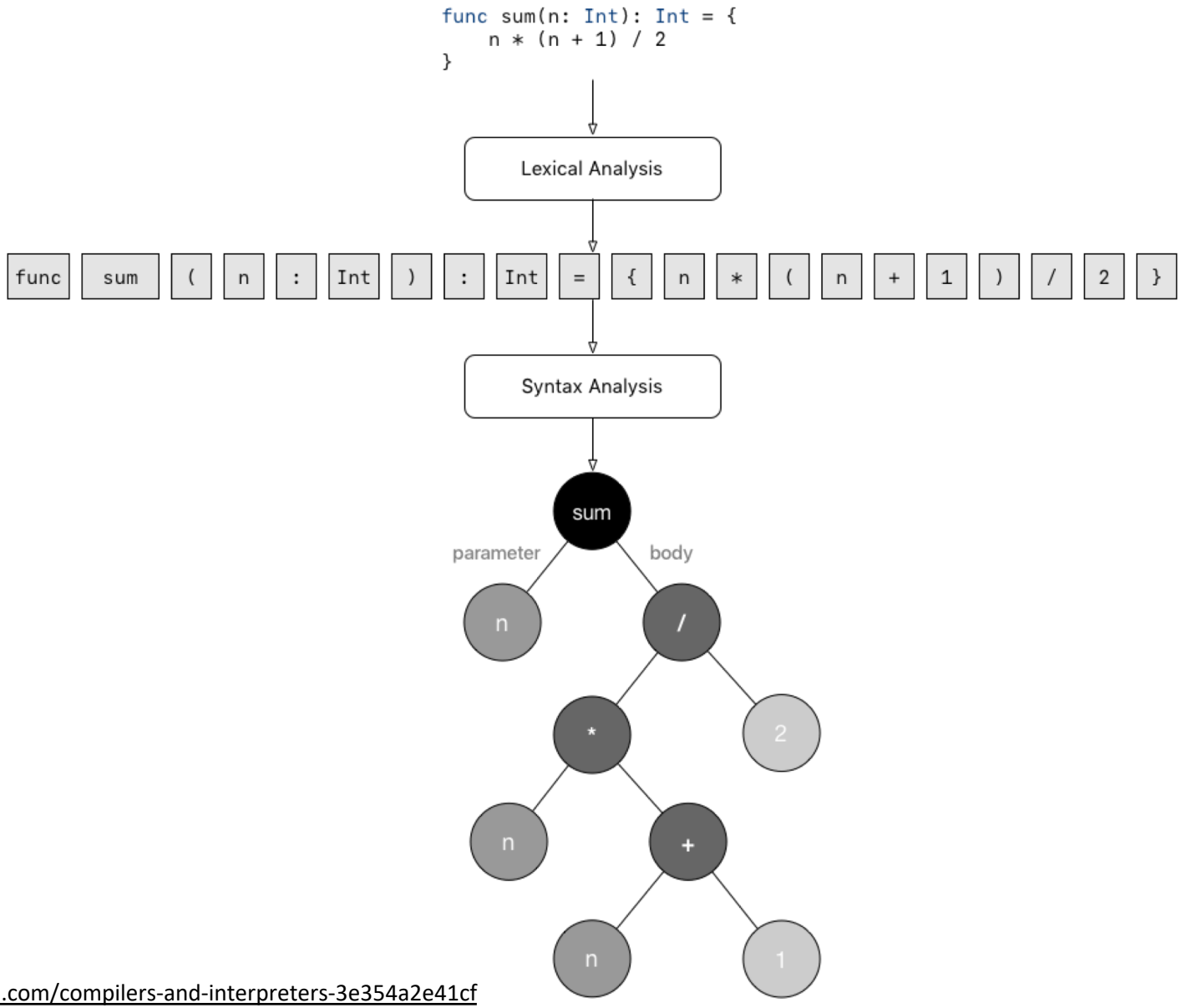
https://hackernoon.com/compilers-and-interpreters-3e354a2e41cf

Compilation – Lexical Analysis



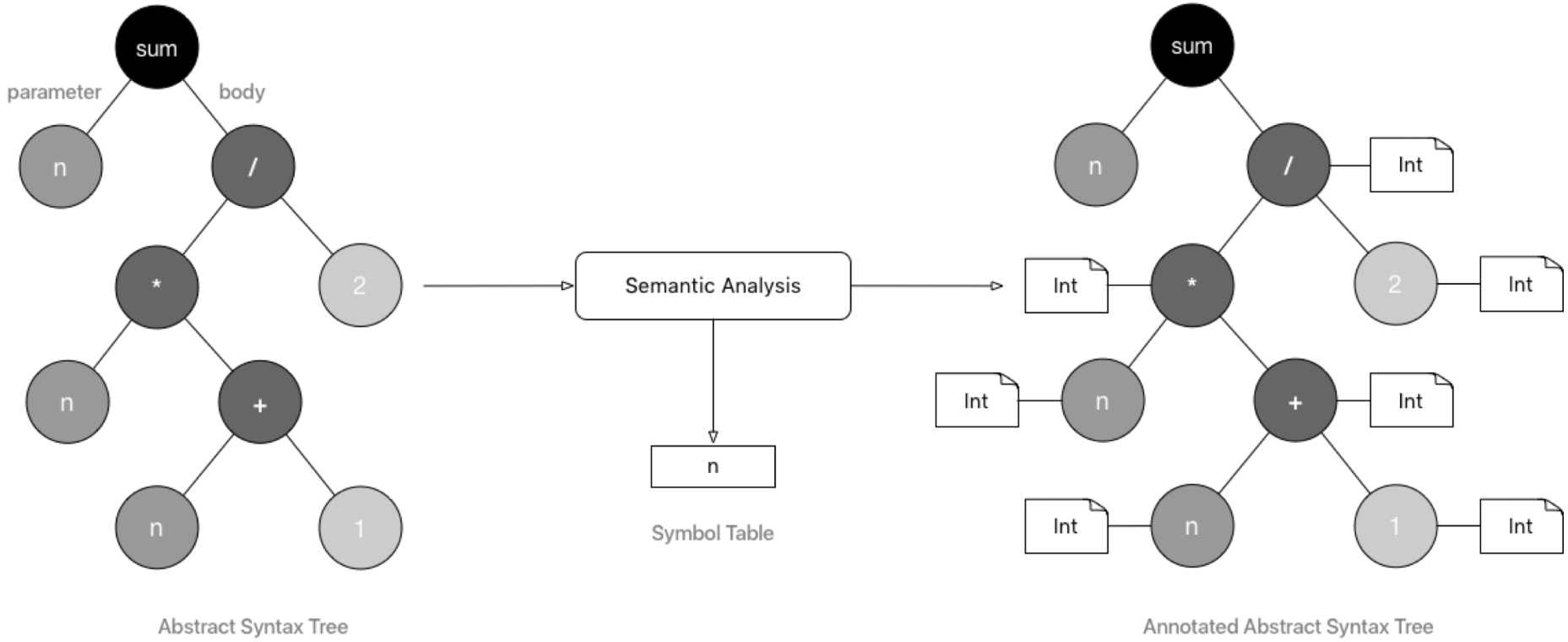
https://hackernoon.com/compilers-and-interpreters-3e354a2e41cf

Compilation – Syntax Analysis



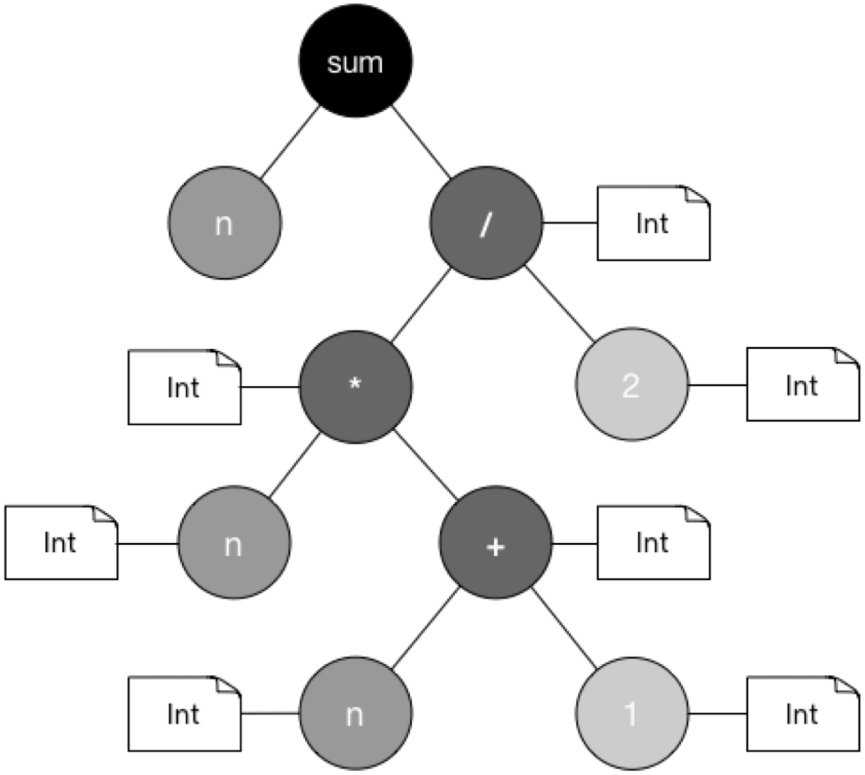
https://hackernoon.com/compilers-and-interpreters-3e354a2e41cf

Compilation – Semantic Analysis



https://hackernoon.com/compilers-and-interpreters-3e354a2e41cf

Compilation – Intermediate Code Generation



Annotated Abstract Syntax Tree

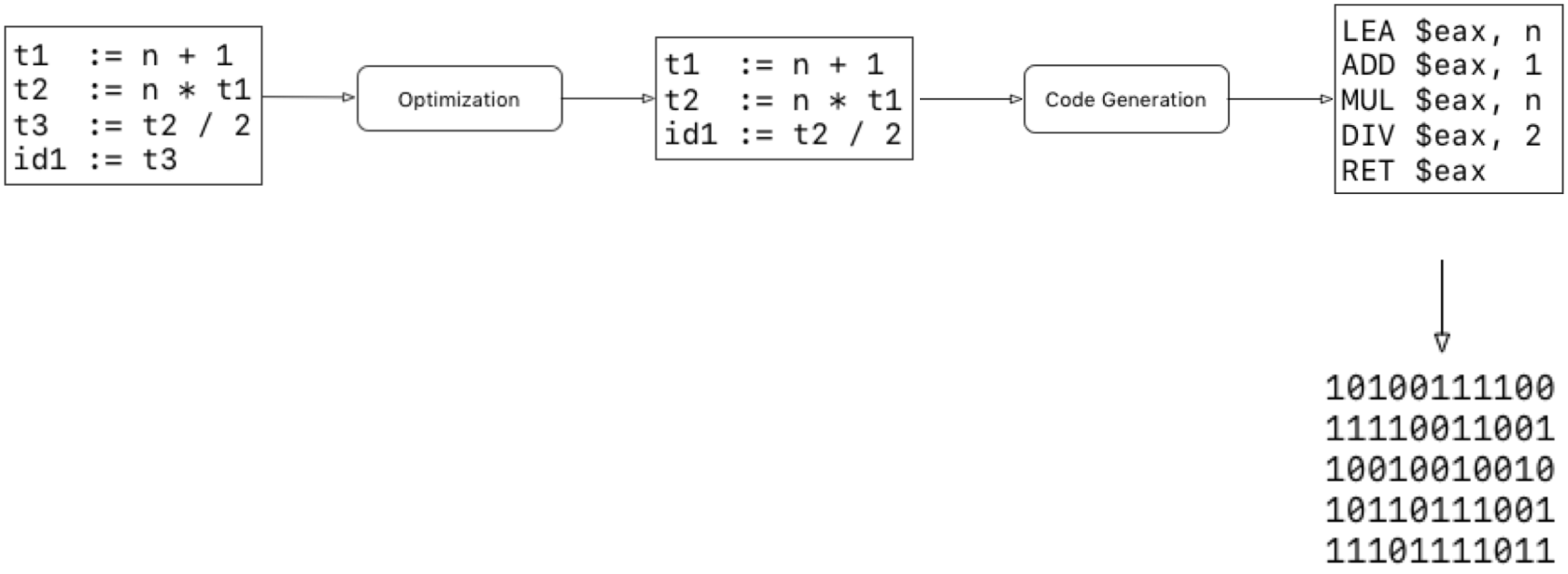
Intermediate Code Generation

```
t1  := n + 1
t2  := n * t1
t3  := t2 / 2
id1 := t3
```

Three-address code

https://hackernoon.com/compilers-and-interpreters-3e354a2e41cf

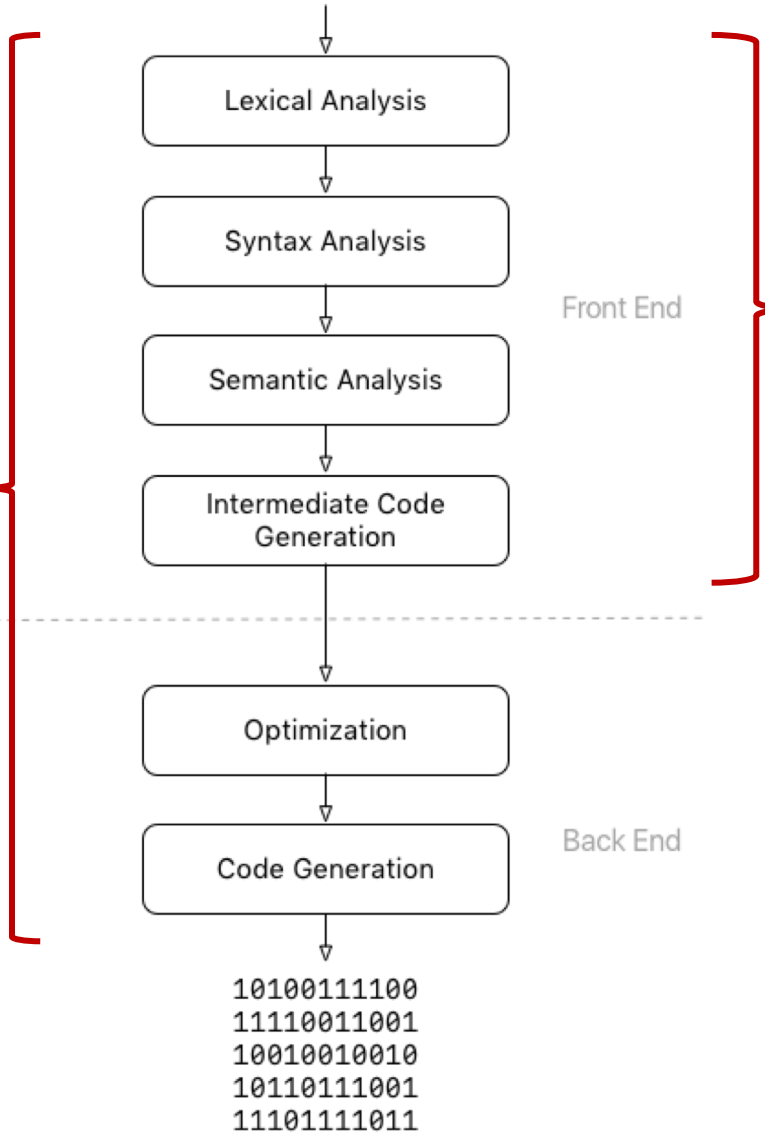
Compilation – Optimization & Code Generation



Explained in good blog post:
https://hackernoon.com/compilers-and-interpreters-3e354a2e41cf

Compiler vs. Interpreter

```
func greet() = {
    Console.println("Hello, World!")
}
```



Compilers execute all parts

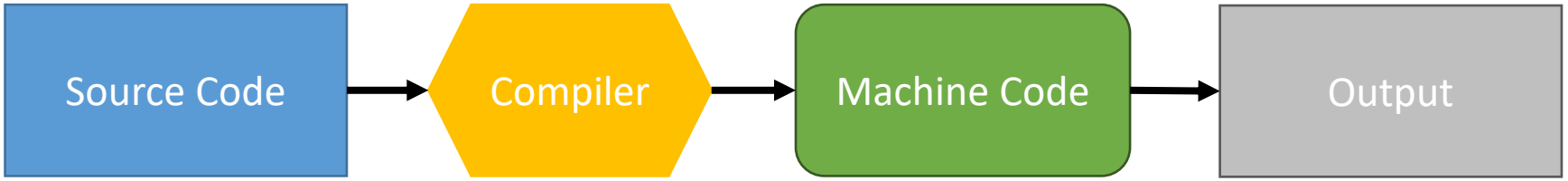
Interpreters only execute this part

```
t1  := n + 1
t2  := n * t1
t3  := t2 / 2
id1 := t3
```

→ No processor specific optimization

Compilers vs. Interpreters

How a compiler works (e.g. C) → Compilation before runtime




```
func greet() = {
    Console.println("Hello, World!")
}
```

```
10100111100
11110011001
10010010010
10110111001
11101111011
```

42

How a interpreter works (e.g. Python) → Compilation at runtime

Hands-On in Pairs!

Write a program in Python that prints:

“Hello World!”

(…or any other message that you would like to spread around)

Execute it in the command line.

Agenda

1. Types of Programming Languages
2. Compilation vs. Interpretation
3. Comparison of C vs. Python
4. Virtual Python Environments using Anaconda

Python vs. C

C

- Compiled language

Python

- Interpreted language

Python vs. C

Python Source Code

```
x = 10
print(x)
```

```
x = “Hello World”
print(x)
```

No explicit type declaration
→ Dynamic typing
→ Type inference during runtime

C Source Code

```
int x;
x = 10;
printf(“%d”, x);
```

```
x = “hello world”
char *y;
y = “hello world”;
printf(“%s”, y);
```

Error

Explicit type declaration
→ Static typing

Python vs. C

Python

- Compilation at runtime
- Simple syntax
- Dynamic typing
- Many built-in functions
- Garbage collection

C

- Compilation before runtime
- More complex syntax
- Static typing
- Code everything from scratch
- Manual memory management

Discuss in Pairs

What are advantages and disadvantages of Python vs. C?

Python vs. C

Python

- Compilation at runtime
- Simple syntax
- Dynamic typing
- Many built-in functions
- Garbarge collection

→ Faster coding

→ Interactive coding and debugging

→ Slower computation

→ No type-safety

C

- Compilation before runtime
- More complex syntax
- Static typing
- No built-in functions
- Manual memory management

→ Slower coding

→ Compile after every change

→ Faster computation

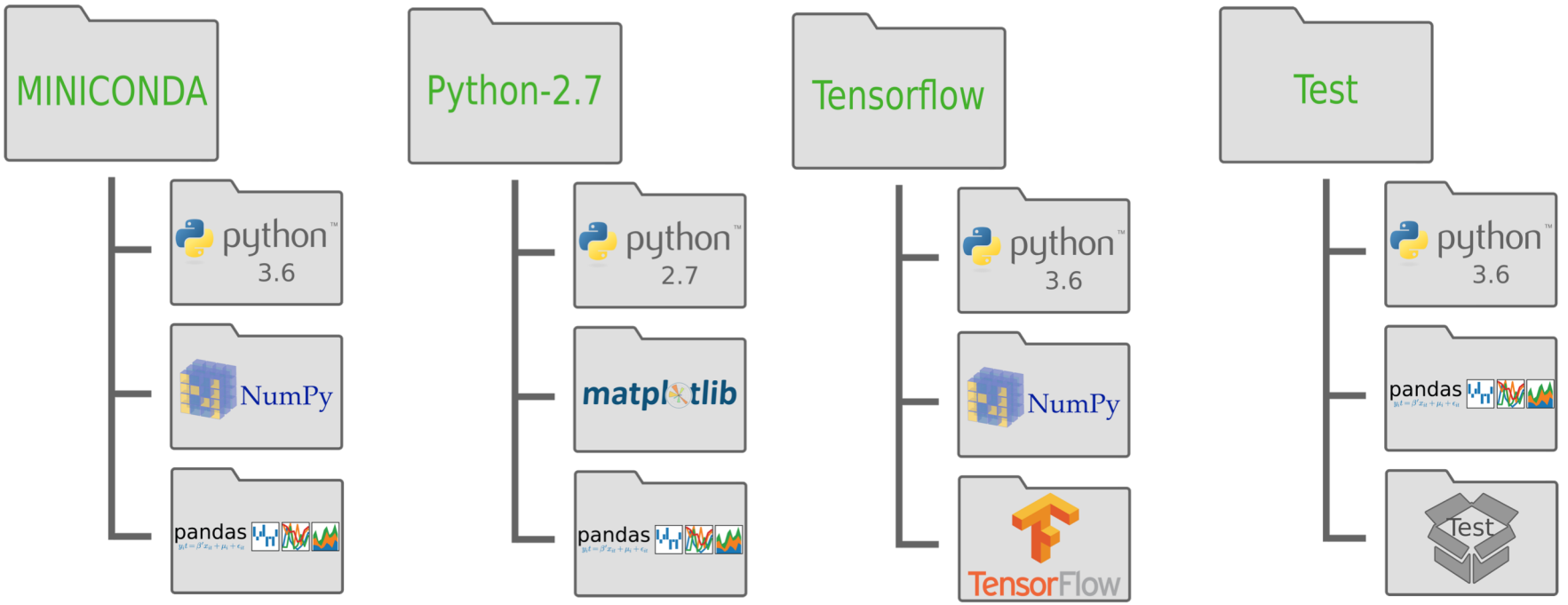
→ Type-safety

Agenda

1. Types of Programming Languages
2. Compilation vs. Interpretation
3. Comparison of C vs. Python
4. Virtual Python Environments using Anaconda

Python Virtual Environments

ANACONDA®



https://python-for-scientists.readthedocs.io/en/latest/_pages/environments.html

Python Environments

Hands-On in Pairs!

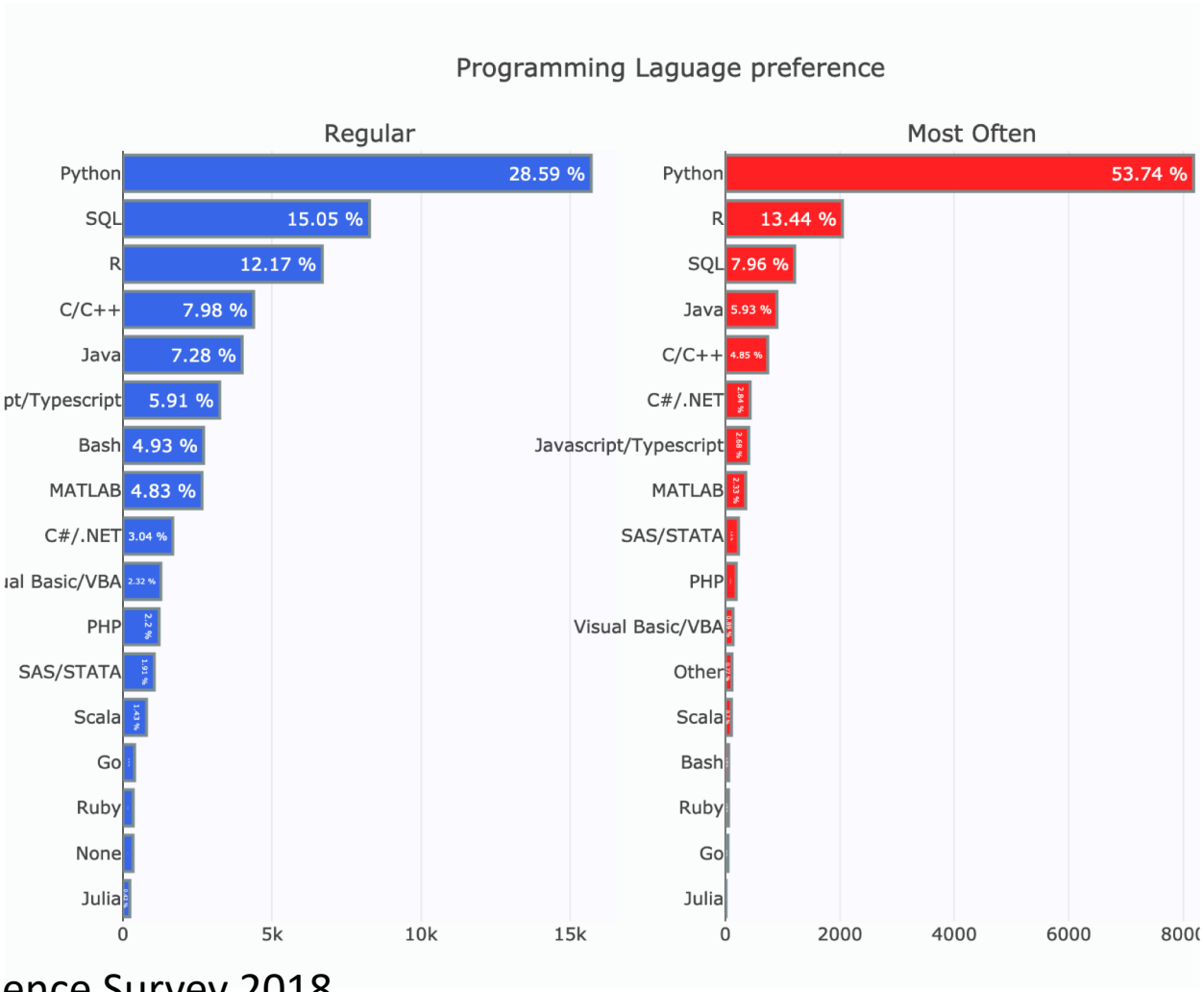
Set up a Python environment using Anaconda.

Summary

- There is no best programming language.
- Python requires more computation time than C, but the simple syntax, built-in functions and garbage collection saves the programmer a lot of time when writing code.
- Combination of Python and C exist
  - Many python functions are implemented in C
  - You can combine Python and C using Cython
- Anaconda allows you to create and manage multiple virtual Python environments, so you can configure Python for different applications.

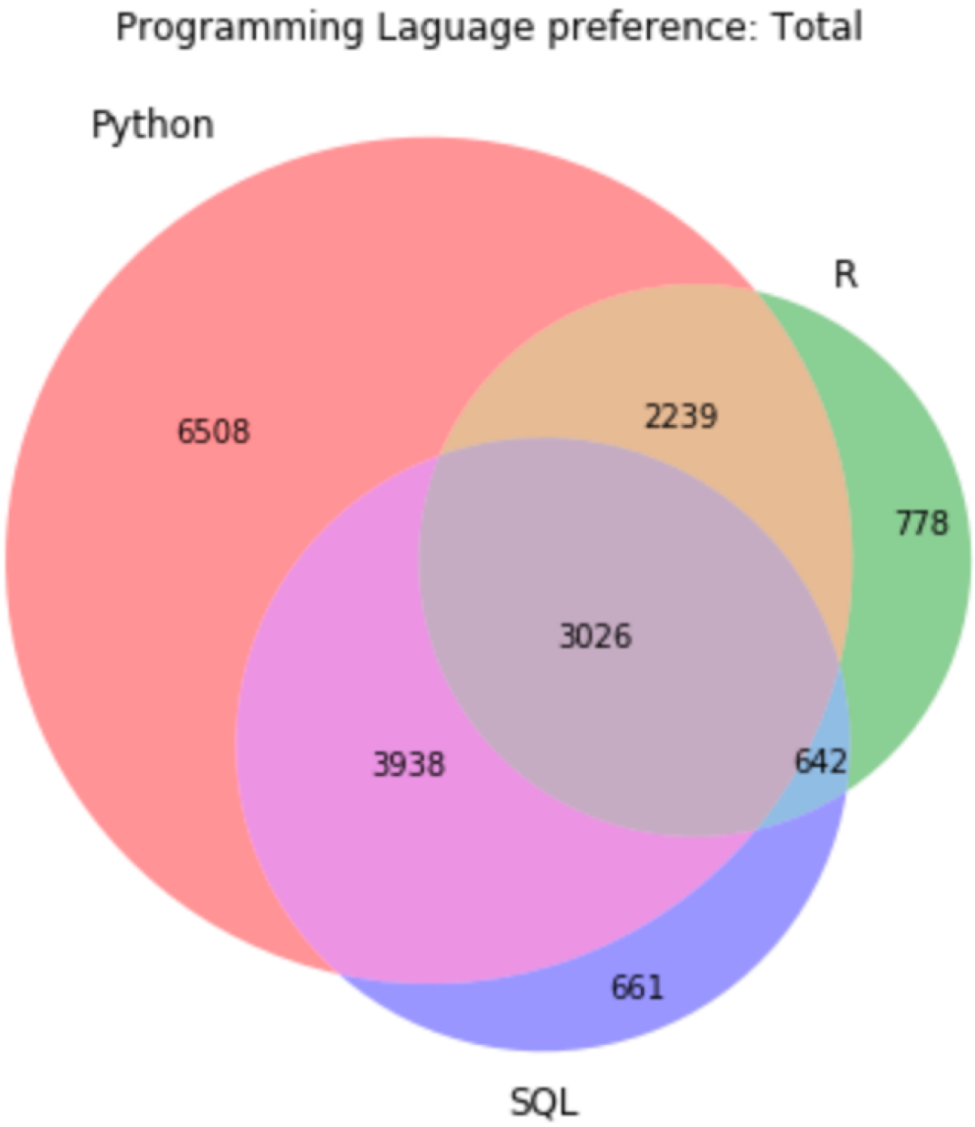
Python for Science

What programming languages do you use on a regular basis? (Select all that apply) - Selected Choice



Kaggle Data Science Survey 2018

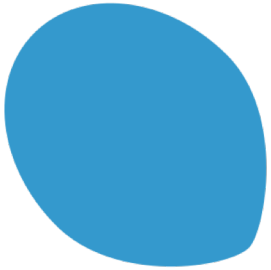
Python for Science



Kaggle Data Science Survey 2018

The world of Scientific Python

SciPy



software carpentry

PyData

Lunch Time!

Resources

Compilation: https://hackernoon.com/compilers-and-interpreters-3e354a2e41cf

Python Interpreter: https://indianpythonista.wordpress.com/2018/01/04/how-python-runs/