Python Introduction
Part 1

Aslak Johansen asjo@mmmi.sdu.dk

April 22, 2026

prosa

Initial Survey

https://PollEv.com/surveys/kFEMdIm57R05xozqmrQan/respond

Part 0:
Motivation

Motivation

Why learn a new programming language?

Motivation

Why learn a new programming language?

1. To keep up with the evolving landscape
2. To learn a new paradigm
3. Because it is a better fit for the problem at hand
4. Because it is fun!

Motivation ▷ To keep up with the evolving landscape

	Past	Present	Future
Scripting:	TCL,Perl	Python+Elixir	Python+Elixir
System:	C	C	Rust/Zig
Concurrency:	Go	Elixir	Elixir+Rust
Presentation:	Postscript	Cairo+SVG+TikZ	Cairo+SVG+TikZ

Motivation ▷ To learn a new paradigm

Motivation ▷ To learn a new paradigm

"A language that doesn't affect the way you think about programming, is not worth knowing"

— *Alan Perlis*

Motivation ▷ To learn a new paradigm

"A language that doesn't affect the way you think about programming, is not worth knowing"

— *Alan Perlis*

"Programming languages have a devious influence: they shape our thinking habits"

— *Edsger W. Dijkstra*

Motivation ▷ To learn a new paradigm

> *"A language that doesn't affect the way you think about programming, is not worth knowing"*
>
> — *Alan Perlis*

> *"Programming languages have a devious influence: they shape our thinking habits"*
>
> — *Edsger W. Dijkstra*

Knowing another paradign represents another option in your toolbox.

Motivation ▷ Because it is a better fit for the problem at hand

Many people know just one or two languages.

Motivation ▷ Because it is a better fit for the problem at hand

Many people know just one or two languages.

What do you do when a new problem comes along where those languages are a bad fit?

1. Use your hammer to dig the hole!
2. Let someone else (who knows how to use a shovel) dig the hole.
3. Learn how to use the right tool (spade?) for the job.

Motivation ▷ Because it is fun!

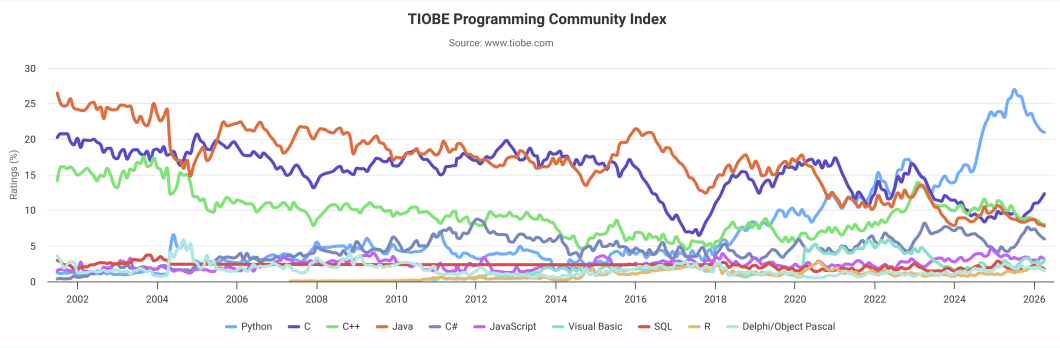
Learning a new language can be a creative experience.

I have a list of problems that I come back to every time I learn a new language

- Sudoku
- Webserver

To learn a new language is to learn a new way of expressing oneself.

Motivation ▷ Popularity



Part 1:
Introduction

Philosophy

- Scripting language.
- Emphasis on code readability.
- Execution speed is not a priority.
- Language simplicity is a priority.
- Readability is a priority.
- Blocks are defined through indentation.

Philosophy

- Scripting language.
- Emphasis on code readability.
- Execution speed is not a priority.
- Language simplicity is a priority.
- Readability is a priority.
- Blocks are defined through indentation.

Zen of Python https://en.wikipedia.org/wiki/Zen_of_Python.

Philosophy

- Scripting language.
- Emphasis on code readability.
- Execution speed is not a priority.
- Language simplicity is a priority.
- Readability is a priority.
- Blocks are defined through indentation.

Zen of Python https://en.wikipedia.org/wiki/Zen_of_Python.

Quoting Monty Python is officially considered good style.

Characteristics

Characteristics

- ***Interpreted:***

Characteristics

- ***Interpreted:***
  - C: compiler + code $\rightarrow$ code $\rightarrow$ execution

Characteristics

- ***Interpreted:***
  - C: compiler + code → code → execution
  - Java: compiler + code → bytecode; bytecode + interpreter → execution

Characteristics

- ***Interpreted:***
  - C: compiler + code → code → execution
  - Java: compiler + code → bytecode; bytecode + interpreter → execution
  - Python: interpreter + code → execution

Characteristics

- ***Interpreted:***
  - C: compiler + code → code → execution
  - Java: compiler + code → bytecode; bytecode + interpreter → execution
  - Python: interpreter + code → execution
- ***Garbage Collected:*** Makes use of automatic memory management (no need to keep tract of when to *free* individual allocations).

Characteristics

- ***Interpreted:***
  - C: compiler + code → code → execution
  - Java: compiler + code → bytecode; bytecode + interpreter → execution
  - Python: interpreter + code → execution
- ***Garbage Collected:*** Makes use of automatic memory management (no need to keep tract of when to *free* individual allocations).
- ***Dynamically Typed:*** Type safety is verified at runtime.

Characteristics

- ***Interpreted:***
  - C: compiler + code → code → execution
  - Java: compiler + code → bytecode; bytecode + interpreter → execution
  - Python: interpreter + code → execution
- ***Garbage Collected:*** Makes use of automatic memory management (no need to keep tract of when to *free* individual allocations).
- ***Dynamically Typed:*** Type safety is verified at runtime.
  - Optional typing: `https://docs.python.org/3/library/typing.html`

Characteristics

- ***Interpreted:***
  - C: compiler + code → code → execution
  - Java: compiler + code → bytecode; bytecode + interpreter → execution
  - Python: interpreter + code → execution
- ***Garbage Collected:*** Makes use of automatic memory management (no need to keep tract of when to *free* individual allocations).
- ***Dynamically Typed:*** Type safety is verified at runtime.
  - Optional typing: `https://docs.python.org/3/library/typing.html`
- ***Late Binding:*** Names are looked up at runtime.

Characteristics

- ***Interpreted:***
  - C: compiler + code → code → execution
  - Java: compiler + code → bytecode; bytecode + interpreter → execution
  - Python: interpreter + code → execution
- ***Garbage Collected:*** Makes use of automatic memory management (no need to keep tract of when to *free* individual allocations).
- ***Dynamically Typed:*** Type safety is verified at runtime.
  - Optional typing: `https://docs.python.org/3/library/typing.html`
- ***Late Binding:*** Names are looked up at runtime.
  - Run-time errors on name lookup.

Characteristics

- ***Interpreted:***
  - C: compiler + code → code → execution
  - Java: compiler + code → bytecode; bytecode + interpreter → execution
  - Python: interpreter + code → execution
- ***Garbage Collected:*** Makes use of automatic memory management (no need to keep tract of when to *free* individual allocations).
- ***Dynamically Typed:*** Type safety is verified at runtime.
  - Optional typing: `https://docs.python.org/3/library/typing.html`
- ***Late Binding:*** Names are looked up at runtime.
  - Run-time errors on name lookup.
- ***High-Level:*** Focus is on serving as a good language for abstract (as in: machine distant) operations.

Good Matches

- String manipulation
- Data transformation
- Batch data processing
- Filesystem traversal
- Quick and dirty coding
- Calculator

Bad Matches

Bad Matches

The use of a global interpreter lock (GIL) makes the language a bad fit for most problems that would benefit from concurrency.

Bad Matches

The use of a global interpreter lock (GIL) makes the language a bad fit for most problems that would benefit from concurrency.

The lack of "compile-time" type checking/safely makes the language a bad fit for any problem that requires robustness or is long-running.

Bad Matches

The use of a global interpreter lock (GIL) makes the language a bad fit for most problems that would benefit from concurrency.

The lack of "compile-time" type checking/safely makes the language a bad fit for any problem that requires robustness or is long-running.

The level of abstraction makes python a bad fit for any computationally heavy operations.

Bad Matches

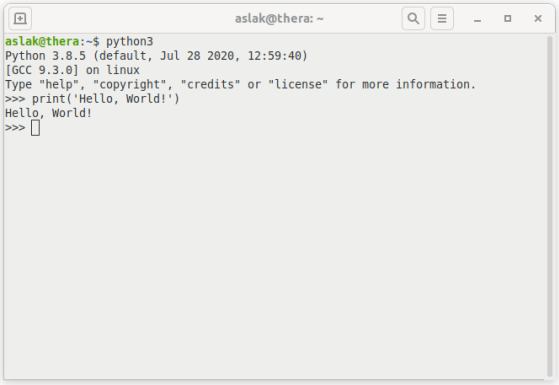
The use of a global interpreter lock (GIL) makes the language a bad fit for most problems that would benefit from concurrency.

The lack of "compile-time" type checking/safely makes the language a bad fit for any problem that requires robustness or is long-running.

The level of abstraction makes python a bad fit for any computationally heavy operations.

- **Note:** Python code can call C, Fortran or CUDA code which can do the heavy lifting. This is what `numpy` does.

Two Modes of Execution



Part 2:
Getting Started

Imports

```
# import everything from module
import os

# import select (comma-separated) names from module
from sys import argv

# import name from module under a different names
from sys import exit as bye

print(os.name)
bye()
```

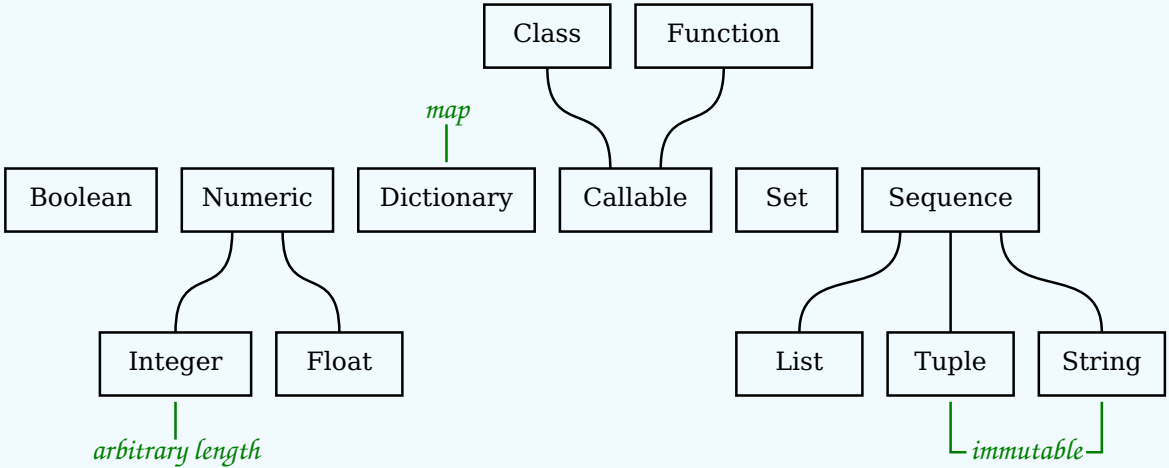
First Steps

```
#!/usr/bin/env python3
import sys

print("Hello, world!")
sys.exit() # this is really not necessary
```

Part 3:
Basic Datatypes and -Structures

Types


Class
Function
map
Boolean
Numeric
Dictionary
Callable
Set
Sequence
Integer
Float
List
Tuple
String
arbitrary length
immutable

Boolean Operators

In python boolean operators are spelled out.

```
if page < pagecount and good_book:
    print('Enjoy!')

while not there:
    print('Are we there yet?')

if finished or not started:
    print('Not much is happening :-(')
```

String Operations

```
>>> s1 = "Alice was beginning to get very tired of sitting by her sister on the bank"
>>> s1
'Alice was beginning to get very tired of sitting by her sister on the bank'
>>> s2 = 'and of having nothing to do'
>>> s2
'and of having nothing to do'
>>> s = s1+", "+s2
>>> s
'Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do'
>>> s = s.replace(',', '')
>>> s
'Alice was beginning to get very tired of sitting by her sister on the bank and of having nothing to do'
>>> words = s.split(' ')
>>> words
['Alice', 'was', 'beginning', 'to', 'get', 'very', 'tired', 'of', 'sitting', 'by', 'her', 'sister', 'on', 'the',
'bank', 'and', 'of', 'having', 'nothing', 'to', 'do']
>>> '%d: %s' % (3, 'March')
'3: March'
>>> '_'.join(words)
'Alice_was_beginning_to_get_very_tired_of_sitting_by_her_sister_on_the_bank_and_of_having_nothing_to_do'
```

Functions

```
def add (a, b, c=0):
    return a+b+c

print(add(1,2,3))
print(add(1,2))

a = add
print(a(1,2))
```

Functions

```
def add (a, b, c=0):
    return a+b+c

print(add(1,2,3))
print(add(1,2))

a = add
print(a(1,2))
```

```
6
3
3
```

Type Introspection

```
>>> t = type(True)
>>> t
<class 'bool'>
>>> type(t)
<class 'type'>
>>> type(bool)
<class 'type'>
>>> t == bool
True
```

Type Introspection

```
>>> def fun(var): return var
...
>>> type(fun)
<class 'function'>
>>> f = fun
>>> type(f)
<class 'function'>
>>> f(1)
1
>>> g = lambda a: a
>>> type(g)
<class 'function'>
>>> g(1)
1
```

Type Conversion

```
>>> int(1.2)
1
>>> float(1)
1.0
>>> int(True)
1
>>> float(True)
1.0
>>> str(True)
'True'
>>> bool(42)
True
```

Part 4:
Flow Control

Branching

```
if len(lines)>0 and len(line[0])>0 and line[0][0]=='#':
    print('First line is a coment')

parts = line.split(' ')
command = parts[0]
if   command=='load':
    load_file()
elif command=='save':
    save_file()
elif command=='quit':
    quit()
else:
    print('Unknown command "'+command+'"')
```

Missing For-Loop

Python does not have a *for* loop.

Missing For-Loop

Python does not have a *for* loop.

Python has a *foreach* loop.

Missing For-Loop

Python does not have a *for* loop.

Python has a *foreach* loop.

Iterating over a list:

```
for line in lines:
    print(line)
```

Missing For-Loop

Python does not have a *for* loop.

Python has a *foreach* loop.

Iterating over a list:

```
for line in lines:
    print(line)
```

Iterating over a list with access to the index:

```
for i in range(len(lines)):
    line = lines[i]
    print(str(i)+': '+line)
```

Generating Ranges of Integers

The `range` function returns a generator for a sequence of integers.

Generating Ranges of Integers

The `range` function returns a generator for a sequence of integers.

```
>>> range(5)
range(0, 5)
>>> list(range(5))
[0, 1, 2, 3, 4]
>>> list(range(1,5))
[1, 2, 3, 4]
>>> list(range(1,5,2))
[1, 3]
>>> for i in range(1,5,2):
...     print(i)
...
1
3
```

Part 5:
Lists

List Operations

```
>>> l = [1,2,3]
>>> l
[1, 2, 3]
>>> l.append(4)
>>> l
[1, 2, 3, 4]
>>> l.extend([7,6,5])
>>> l
[1, 2, 3, 4, 7, 6, 5]
>>> sorted(l)
[1, 2, 3, 4, 5, 6, 7]
>>> l
[1, 2, 3, 4, 7, 6, 5]
>>> l.sort()
>>> l
[1, 2, 3, 4, 5, 6, 7]
```

```
>>> len(l)
7
>>> l[2], l[-1]
(3, 7)
>>> l[2:]
[3, 4, 5, 6, 7]
>>> l[2:4]
[3, 4]
>>> l[:4]
[1, 2, 3, 4]
>>> 4 in l, 42 in l
(True, False)
```

Part 6:
Dictionaries

Dictionary Operations [1/2]

```
>>> {}
{}
>>> d = {'jan': 1, 'feb': 2, 'mar': 3}
>>> d
{'jan': 1, 'feb': 2, 'mar': 3}
>>> d['jan']
1
>>> d['apr'] = 4
>>> d
{'jan': 1, 'feb': 2, 'mar': 3, 'apr': 4}
>>> d['list'] = [1,2,3]
>>> d
{'jan': 1, 'feb': 2, 'mar': 3, 'apr': 4, 'list': [1, 2, 3]}
>>> 'jan' in d
True
>>> 'may' in d
False
```

Dictionary Operations [2/2]

```
>>> d.keys()
dict_keys(['jan', 'feb', 'mar', 'apr', 'list'])
>>> list(d.keys())
['jan', 'feb', 'mar', 'apr', 'list']
>>> for key in d: print(key)
...
jan
feb
mar
apr
list
>>> del(d['feb'])
>>> d
{'jan': 1, 'mar': 3, 'apr': 4, 'list': [1, 2, 3]}
```

Part 7:
Strings

Strings: Basic Operations

```
>>> initial = ' once upon a time  '
>>> initial
' once upon a time  '
>>> len(initial)
19
>>> stripped = initial.strip()
>>> stripped
'once upon a time'
>>> words = stripped.split(' ')
>>> words
['once', 'upon', 'a', 'time']
>>> words[1], stripped[1]
('upon', 'n')
>>> joined = '_'.join(words)
>>> joined
'once_upon_a_time'
```

Part 8:
Object-Orientation

Object-Orientation ▷ Properties

```
class Person:
  name = "Nobody"

p = Person()

print(p.name)
```

Object-Orientation ▷ Methods

```
class Person:
  name = "Nobody"

  def set_name(self, name):
    self.name = name

  def get_name(self):
    return self.name

p = Person()
p.set_name("Aslak")
print(p.get_name())
```

Object-Orientation ▷ Constructors

```
class Person:
  name = "Nobody"

  def __init__(self, name):
    self.name = name

  def set_name(self, name):
    self.name = name

  def get_name(self):
    return self.name

p = Person("Aslak")
print(p.get_name())
```

Object-Orientation ▷ Inheritance

```
class Animal:
  alive = False

class Person (Animal):
  name = "Nobody"

  def __init__(self, name):
    self.name = name
    self.alive = True

p = Person("Aslak")
print(p.alive)
```

Object-Orientation ▷ Encapsulation

```
class Person:
  __name = "Nobody"

  def set_name(self, name):
    self.__name = name

  def get_name(self):
    return self.__name

p = Person()
p.set_name("Aslak")
print(p.get_name())
print(p.__name)
```

Object-Orientation ▷ Abstract Classes

```
from abc import ABC, abstractmethod

class Shape(ABC):
  @abstractmethod
  def area(self):
      pass

class Square(Shape):
  height = -1
  width = -1

  def __init__(self, height, width):
    self.height = height
    self.width = width

  def area(self):
    return self.height * self.width

s = Square(2, 3)
print(s.area())
```

Object-Orientation ▷ Interfaces

Object-Orientation ▷ Interfaces

Python does not have a notion of interfaces.

Object-Orientation ▷ Interfaces

Python does not have a notion of interfaces.

The `abc` module does not have a notion of interfaces.

Object-Orientation ▷ Interfaces

Python does not have a notion of interfaces.

The `abc` module does not have a notion of interfaces.

Python does support multiple inheritance and we have abstract classes.

Object-Orientation ▷ Interfaces

Python does not have a notion of interfaces.

The `abc` module does not have a notion of interfaces.

Python does support multiple inheritance and we have abstract classes.

That is enough.

Object-Orientation ▷ Dynamics

```
class Person:
    def __init__ (self, name):
        self.name = name

    def get_name (self): return self.name
    def set_name (self, name):
        self.name = name

def printname (self):
    print(self.name)

p = Person('Aslak')
print(p.get_name())

setattr(Person, 'printname', printname)
p.printname()
```

Part 9:
Higher-Order Functions

Higher-Order Functions

```
>>> l = [-17,2,5,-4,4,7,-3,-1,9,1]
>>> incr = lambda v: v+1
>>> map(incr, l)
<map object at 0x7f33c8440b20>
>>> list(map(incr, l))
[-16, 3, 6, -3, 5, 8, -2, 0, 10, 2]
>>> pos = lambda v: v>=0
>>> filter(pos, l)
<filter object at 0x7f33c8440b20>
>>> list(filter(pos, l))
[2, 5, 4, 7, 9, 1]
>>> list(map(incr, filter(pos, l)))
[3, 6, 5, 8, 10, 2]
```

Higher-Order Functions

```
>>> l = [-17,2,5,-4,4,7,-3,-1,9,1]
>>> incr = lambda v: v+1
>>> map(incr, l)
<map object at 0x7f33c8440b20>
>>> list(map(incr, l))
[-16, 3, 6, -3, 5, 8, -2, 0, 10, 2]
>>> pos = lambda v: v>=0
>>> filter(pos, l)
<filter object at 0x7f33c8440b20>
>>> list(filter(pos, l))
[2, 5, 4, 7, 9, 1]
>>> list(map(incr, filter(pos, l)))
[3, 6, 5, 8, 10, 2]
```

These functions create generators that can be manifested as a list by wrapping in `list()`.

Part 10:
Working in Modules

`awesome.py` (the module)

```
def add (a, b):
    return a+b
```

Working in Modules ▷ Own Modules

`awesome.py` (the module)

```
def add (a, b):
    return a+b
```

`main.py` (the consumer of the module)

```
import awesome
print(awesome.add(1,2))
```

awesome.py (the module)

```
def add (a, b):
    return a+b
```

main.py (the consumer of the module)

```
import awesome
print(awesome.add(1,2))
```

Result of execution

```
$ python3 main.py
3
```

Working in Modules ▷ Select Modules

Mathmatics: https://docs.python.org/3/library/math.html

Random numbers: https://docs.python.org/3/library/random.html

Part 11:
Pitfalls

Mutable Default Arguments

```
def incr (increment=1, data=[], newvalues=[]):
    for value in newvalues:
        data.append(value)
    for i in range(len(data)):
        data[i] += increment
    return data

print(incr(newvalues=[1]))
print(incr(newvalues=[1]))
```

Mutable Default Arguments

```
def incr (increment=1, data=[], newvalues=[]):
    for value in newvalues:
        data.append(value)
    for i in range(len(data)):
        data[i] += increment
    return data

print(incr(newvalues=[1]))
print(incr(newvalues=[1]))
```

```
$ python3 mutabledefaults.py
[2]
[3, 2]
```

Scope and Globals

```
a = 42
b = 56

def fun (value):
    global b

    a = value
    b = value
    print(a, b)

fun(-1)
print(a, b)
```

Scope and Globals

```
a = 42
b = 56

def fun (value):
    global b

    a = value
    b = value
    print(a, b)

fun(-1)
print(a, b)
```

```
$ python3 globals.py
-1 -1
42 -1
```

Part 12:
Exercises

Exercises ▷ Level 1

Given a book `price` (e.g., 599.95), a `budget` (e.g., 1000.00) and a truth value for whether the book is `required_reading` (e.g., true), a purchase should be made if the book is required reading and within budget. Write a script that determines this, and verify that it works in all cases.

Declare two variables to represent a month and a day of that month. If the combined date is December 24th, print out "It is Christmas!".

Declare a list of integers. Print out the largest element.

Write a script that implements a "factorial function" and tests it. The factorial function is defined as:

$$\begin{aligned} \mathrm{fac}(1) &= 1 \\ \mathrm{fac}(i) &= i \cdot \mathrm{fac}(i-1) \end{aligned}$$

Exercises ▷ Level 2

Write a script that prints out a table converting from °F to °C. The formula is: $C = (F - 32) \cdot \frac{5}{9}$

Write a script that calculates and prints out the area ($\pi \cdot r^2$) of three circles with radiuses of 1, 3 and 5.

Write a script that, based on a list of daily temperatures (e.g., [21.5, 23.7, 19.6, 22.5, 25.3, 21.7, 18.9]), calculates all the daily differences. The first such difference is $23.7 - 21.5 = 2.2$.

Produce, and print out a 10×10 multiplication table.

Write a script that calculates and prints out the roots of a quadratic polynomial with concrete values for *a*, *b* and *c*. It should have a `determinant` function that calculates the determinant, and a `roots` function that returns a list of roots. Use Wikipedia for definitions.

Exercises ▷ Level 3 ▷ Sieve of Eratosthenes

The Sieve of Eratosthenes is a fast and simple way of calculating all prime numbers below some given limit:
https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes

Exercise:

1. Make an implementation of a trivial prime generator.
2. Make an implementation of the Sieve of Eratosthenes.
3. Look up time documentation for `time.time`:
   https://docs.python.org/3/library/time.html#time.time
4. Benchmark the two implementations.

Exercises ▷ Level 3 ▷ Sūdoku

Sūdoku is a type of puzzle: https://en.wikipedia.org/wiki/Sudoku

Exercise:

1. Come up with a data structure that represents a Sūdoku puzzle.
2. Write a solver that fills in the blanks.
3. Write a verifier that tests whether any of the rules have been broken.
4. Write a generator that produces a new puzzle.

Exercises ▷ Level 3 ▷ Traveling Salesman Problem [1/3]

A traveling salesman needs to visit a number of cities (and end up in the first city). This means that she needs to travel in a cycle covering all cities. She wants to minimize the total distance traveled.
https://en.wikipedia.org/wiki/Travelling_salesman_problem

Exercise:

1. The following two slides contains the distances between the 15 largest cities in Denmark as C code. Convert it to python.
2. Write code that solves the traveling salesman problem on these data. Start with the simplest possible solution that you can think of, and then refine it.
3. How do you argue that your result is correct?

Exercises ▷ Level 3 ▷ Traveling Salesman Problem [2/3]

```c
#define CITY_COUNT (15)

char* citynames[CITY_COUNT] = {
    "Esbjerg",
    "Helsingør",
    "Herning",
    "Horsens",
    "Kolding",
    "København",
    "Næstved",
    "Odense",
    "Randers",
    "Roskilde",
    "Silkeborg",
    "Vejle",
    "Viborg",
    "Aalborg",
    "Århus",
};
```

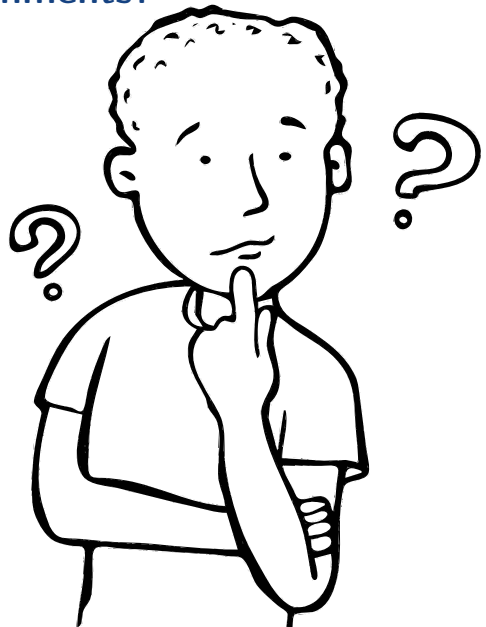
Exercises ▷ Level 3 ▷ Traveling Salesman Problem [3/3]

```
int distances[CITY_COUNT][CITY_COUNT] = {
    {  0, 269,  82,  98,  65, 260, 211, 123, 149, 230, 105,  74, 126, 198, 134},
    {269,   0, 226, 173, 206,  40, 105, 157, 166,  55, 191, 196, 207, 200, 150},
    { 82, 226,   0,  63,  79, 230, 202, 121,  76, 202,  36,  59,  45, 117,  77},
    { 98, 173,  63,   0,  48, 172, 139,  62,  68, 142,  40,  26,  75, 132,  40},
    { 65, 206,  79,  48,   0, 196, 148,  59, 114, 165,  77,  25, 110, 176,  88},
    {260,  40, 230, 172, 196,   0,  71, 141, 180,  31, 196, 190, 218, 224, 156},
    {211, 105, 202, 139, 148,  71,   0,  89, 174,  50, 175, 150, 204, 232, 142},
    {123, 157, 121,  62,  59, 141,  89,   0, 121, 110, 102,  64, 136, 186,  86},
    {149, 166,  76,  68, 114, 180, 174, 121,   0, 156,  44,  90,  42,  65,  36},
    {230,  55, 202, 142, 165,  31,  50, 110, 156,   0, 169, 160, 193, 206, 130},
    {105, 191,  36,  40,  77, 196, 175, 102,  44, 169,   0,  53,  35,  99,  41},
    { 74, 196,  59,  26,  25, 190, 150,  64,  90, 160,  53,   0,  86, 151,  65},
    {126, 207,  45,  75, 110, 218, 204, 136,  42, 193,  35,  86,   0,  72,  63},
    {198, 200, 117, 132, 176, 224, 232, 186,  65, 206,  99, 151,  72,   0, 101},
    {134, 150,  77,  40,  88, 156, 142,  86,  36, 130,  41,  65,  63, 101,   0},
};
```

Where to Find

https://python.asjo.dk

Questions and Comments?



https://openclipart.org/detail/238687/boy-thinking-of-question