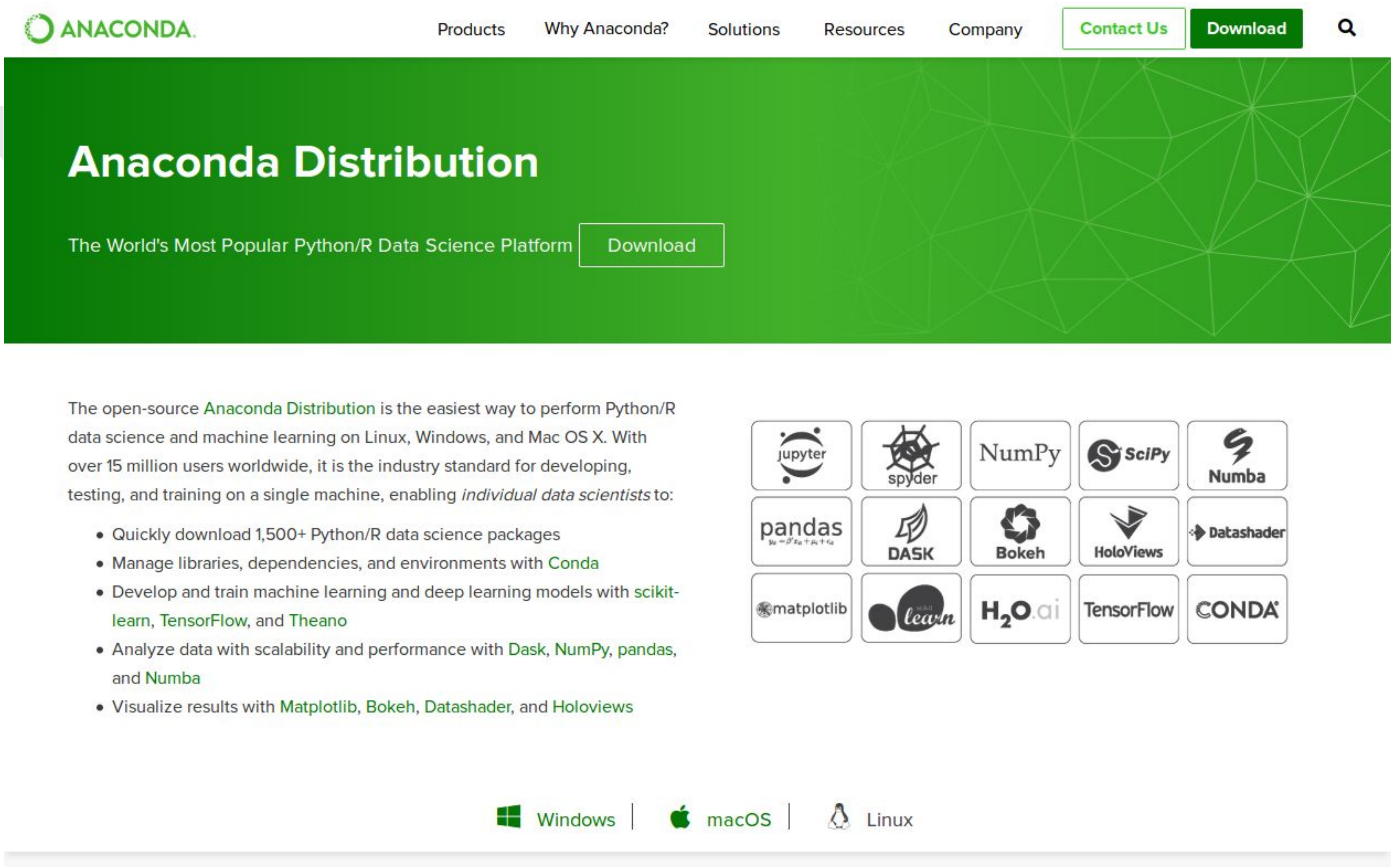
Introduction to Python

SIT training program

What is Python?

- Open source software
  - Open Source Initiative (OSI) organisation
- Interpreted language, not compiled
- High level language
- Clean syntax
- Object Oriented Programming language (OOP)
- Differences with other PLs: R
  - Syntax, speed
- Programming environments
  - Visual Studio Code
- **Anaconda (Python 3.7)**
  - **Jupyter Notebook**



ANACONDA. Products Why Anaconda? Solutions Resources Company Contact Us Download

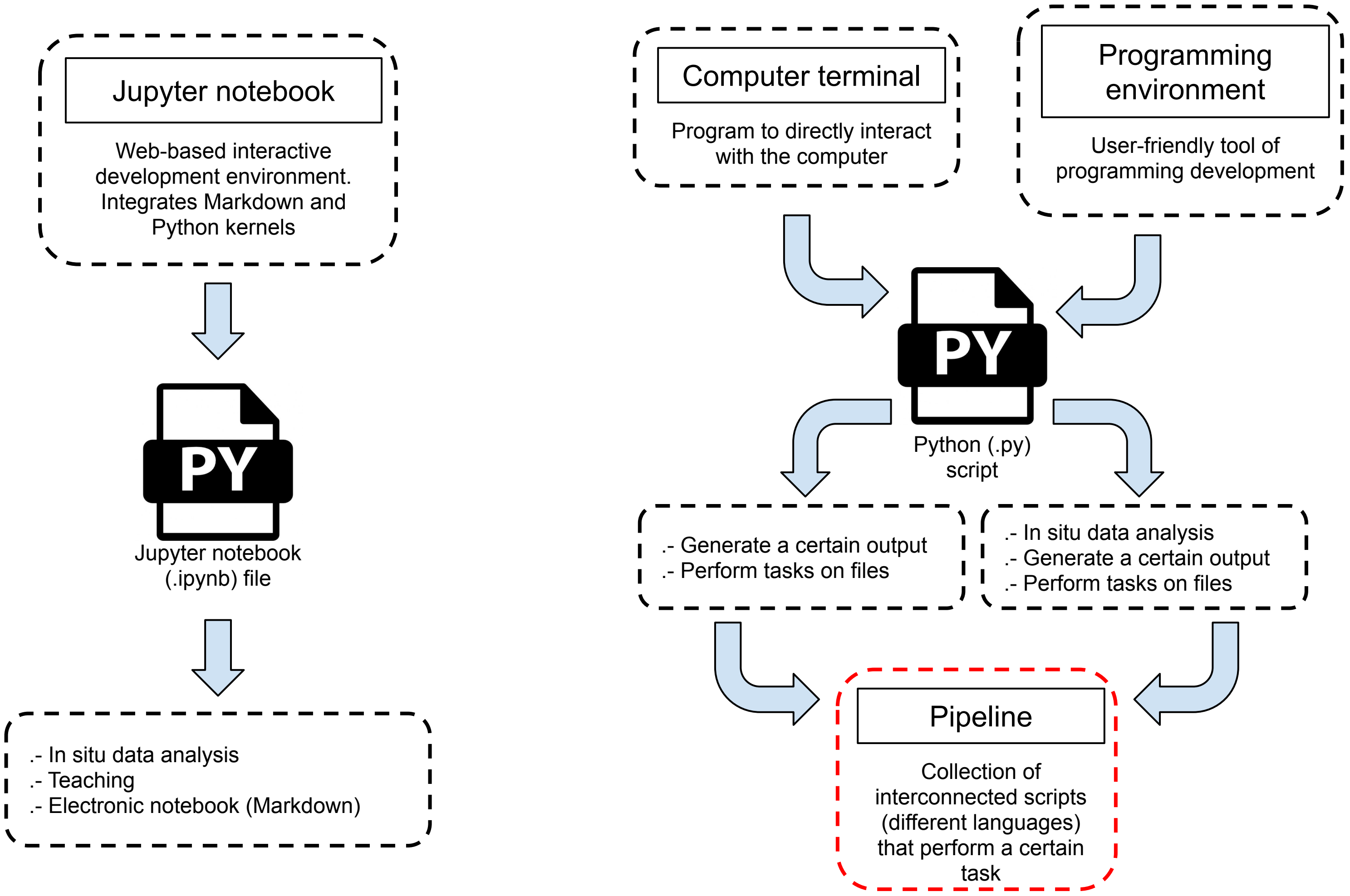
Anaconda Distribution

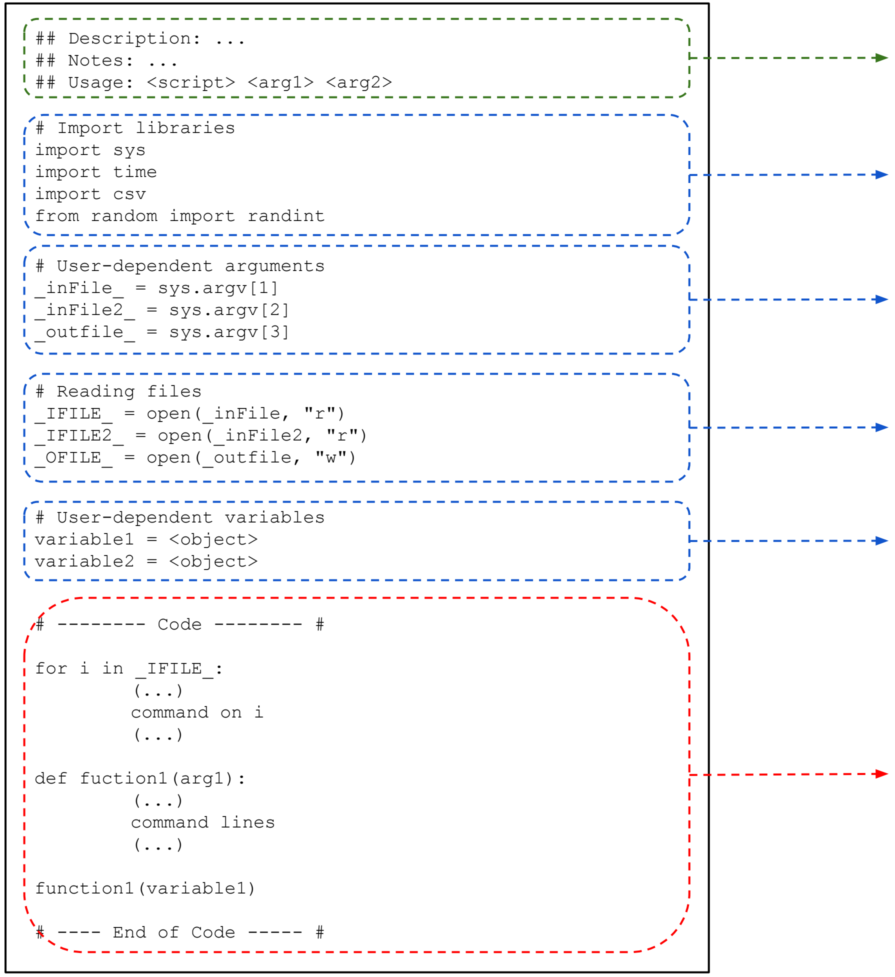
The World's Most Popular Python/R Data Science Platform Download

The open-source Anaconda Distribution is the easiest way to perform Python/R data science and machine learning on Linux, Windows, and Mac OS X. With over 15 million users worldwide, it is the industry standard for developing, testing, and training on a single machine, enabling *individual data scientists* to:

- Quickly download 1,500+ Python/R data science packages
- Manage libraries, dependencies, and environments with Conda
- Develop and train machine learning and deep learning models with scikit-learn, TensorFlow, and Theano
- Analyze data with scalability and performance with Dask, NumPy, pandas, and Numba
- Visualize results with Matplotlib, Bokeh, Datashader, and Holoviews

Windows | macOS | Linux


Jupyter notebook
Web-based interactive development environment. Integrates Markdown and Python kernels
PY
Jupyter notebook (.ipynb) file
.- In situ data analysis
.- Teaching
.- Electronic notebook (Markdown)
Computer terminal
Program to directly interact with the computer
Programming environment
User-friendly tool of programming development
PY
Python (.py) script
.- Generate a certain output
.- Perform tasks on files
.- In situ data analysis
.- Generate a certain output
.- Perform tasks on files
Pipeline
Collection of interconnected scripts (different languages) that perform a certain task



```
## Description: ...
## Notes: ...
## Usage: <script> <arg1> <arg2>

# Import libraries
import sys
import time
import csv
from random import randint

# User-dependent arguments
_inFile_ = sys.argv[1]
_inFile2_ = sys.argv[2]
_outfile_ = sys.argv[3]

# Reading files
_IFILE_ = open(_inFile, "r")
_IFILE2_ = open(_inFile2, "r")
_OFILE_ = open(_outfile, "w")

# User-dependent variables
variable1 = <object>
variable2 = <object>

# -------- Code -------- #

for i in _IFILE_:
        (...)
        command on i
        (...)

def fuction1(arg1):
        (...)
        command lines
        (...)

function1(variable1)

# ---- End of Code ----- #
```

.- Description of the script (what is the purpose of this program?)
.- Notes (special features of this program)
.- Usage (how to run this program in the command line)

.- Import libraries and define library aliases

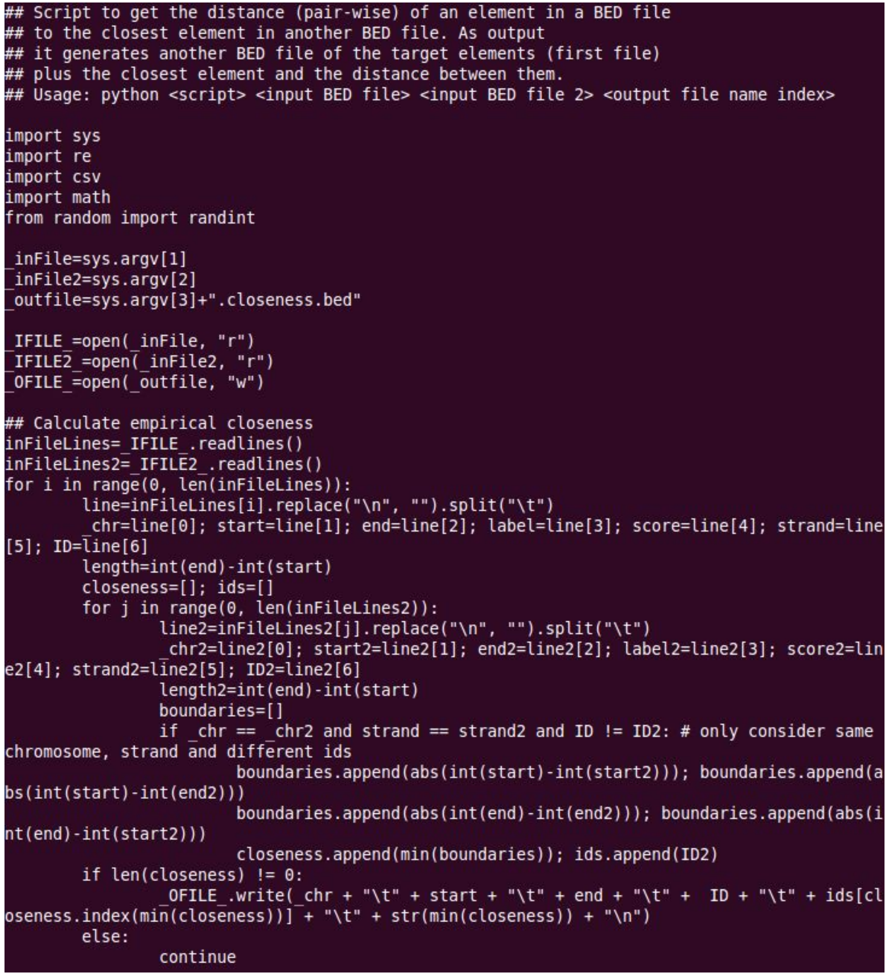
.- Access to parsed arguments (sys library, *argv* function)
.- Store arguments in variables

.- Read files through arguments (file paths). Here we use the open function that accept as arguments the path file and the mode ("r" = reading, "w" = writing.

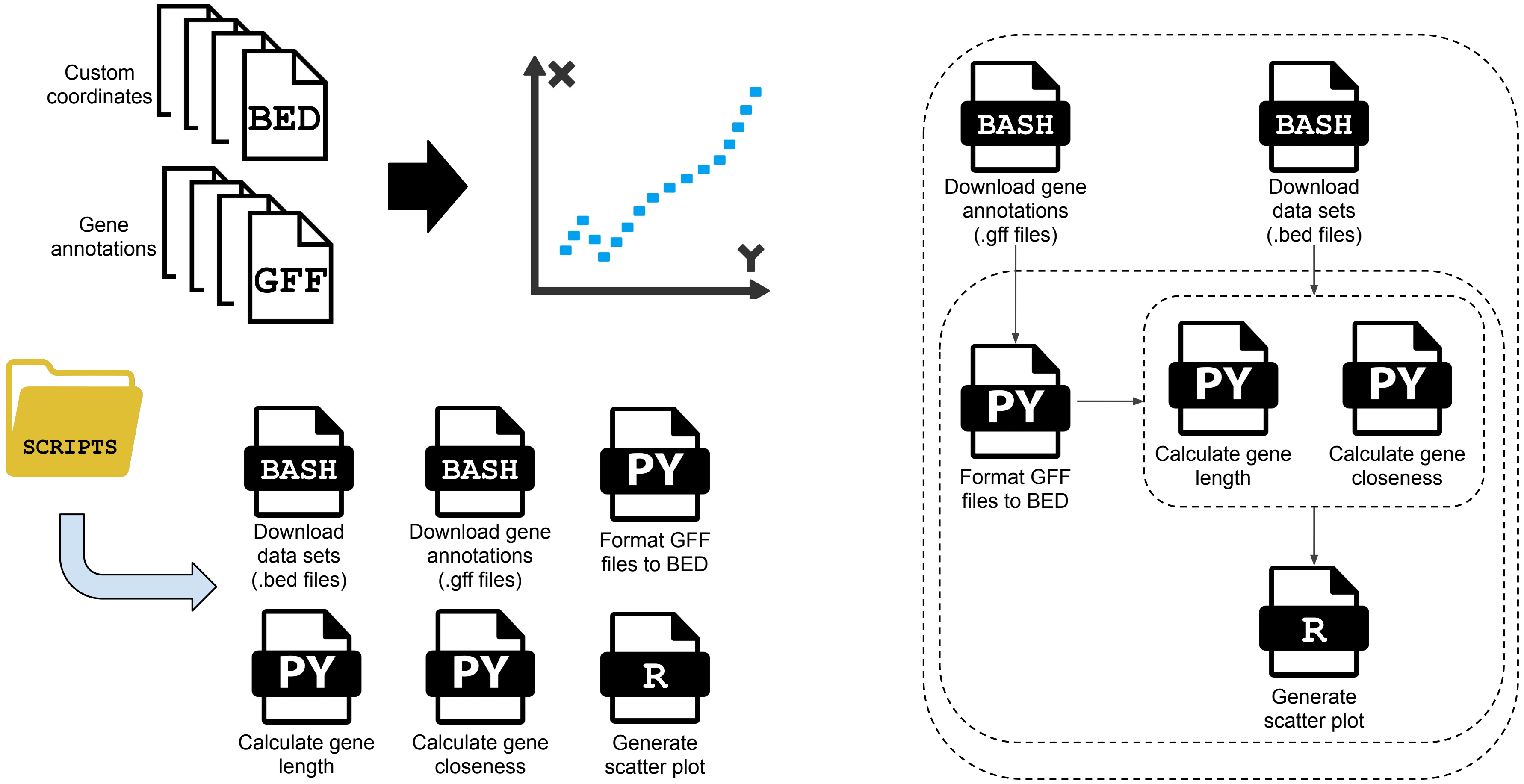
.- If we use pre-defined variables, we create them before the main code.

.- Main code body. Here we write the code of the program itself.

```
## Description: ...
## Notes: ...
## Usage: <script> <arg1> <arg2>

# Import libraries
import sys
import time
import csv
from random import randint


# User-dependent arguments
_inFile_ = sys.argv[1]
_inFile2_ = sys.argv[2]
_outfile_ = sys.argv[3]


# Reading files
_IFILE_ = open(_inFile, "r")
_IFILE2_ = open(_inFile2, "r")
_OFILE_ = open(_outfile, "w")


# User-dependent variables
variable1 = <object>
variable2 = <object>


# -------- Code -------- #

for i in _IFILE_:
         (...)
         command on i
         (...)

def fuction1(arg1):
         (...)
         command lines
         (...)

function1(variable1)

# ---- End of Code ----- #
```



```
## Script to get the distance (pair-wise) of an element in a BED file
## to the closest element in another BED file. As output
## it generates another BED file of the target elements (first file)
## plus the closest element and the distance between them.
## Usage: python <script> <input BED file> <input BED file 2> <output file name index>

import sys
import re
import csv
import math
from random import randint

_inFile=sys.argv[1]
_inFile2=sys.argv[2]
_outfile=sys.argv[3]+".closeness.bed"

_IFILE_=open(_inFile, "r")
_IFILE2_=open(_inFile2, "r")
_OFILE_=open(_outfile, "w")

## Calculate empirical closeness
inFileLines=_IFILE_.readlines()
inFileLines2=_IFILE2_.readlines()
for i in range(0, len(inFileLines)):
        line=inFileLines[i].replace("\n", "").split("\t")
        _chr=line[0]; start=line[1]; end=line[2]; label=line[3]; score=line[4]; strand=line
[5]; ID=line[6]
        length=int(end)-int(start)
        closeness=[]; ids=[]
        for j in range(0, len(inFileLines2)):
                line2=inFileLines2[j].replace("\n", "").split("\t")
                _chr2=line2[0]; start2=line2[1]; end2=line2[2]; label2=line2[3]; score2=lin
e2[4]; strand2=line2[5]; ID2=line2[6]
                length2=int(end)-int(start)
                boundaries=[]
                if _chr == _chr2 and strand == strand2 and ID != ID2: # only consider same
chromosome, strand and different ids
                        boundaries.append(abs(int(start)-int(start2))); boundaries.append(a
bs(int(start)-int(end2)))
                        boundaries.append(abs(int(end)-int(end2))); boundaries.append(abs(i
nt(end)-int(start2)))
                        closeness.append(min(boundaries)); ids.append(ID2)
        if len(closeness) != 0:
                _OFILE_.write(_chr + "\t" + start + "\t" + end + "\t" +  ID + "\t" + ids[cl
oseness.index(min(closeness))] + "\t" + str(min(closeness)) + "\n")
        else:
                continue
```


Custom coordinates
BED
Gene annotations
GFF
X
Y
SCRIPTS
BASH
Download data sets (.bed files)
BASH
Download gene annotations (.gff files)
PY
Format GFF files to BED
PY
Calculate gene length
PY
Calculate gene closeness
R
Generate scatter plot
BASH
Download gene annotations (.gff files)
BASH
Download data sets (.bed files)
PY
Format GFF files to BED
PY
Calculate gene length
PY
Calculate gene closeness
R
Generate scatter plot

How to solve a problem in programming?

1. What do you want to solve? **Understand** the goal of the problem, the final result.
2. In Object-Oriented-Programming this means: understand in **what kind of object** you want to store the result.
3. In order to reach that result, there is a process that goes from the input (e. arguments) to the output (result). Try to divide that process in **simple steps**. If you need to create smaller functions within the main one, do it.
4. Write down these steps: **pseudocode**. This is a way to describe each step: what each step does and how.
5. **Translate** the pseudocode into real code, line by line.
6. **Comment** your code!
7. **Try** your code with simple examples

Example 1

Exercise: Can you create a function that identifies the variant position between these two DNA sequences of the same length?

Sequence 1: CGATCGATGCTA

Sequence 2: CGATCGAGGCTA

Report the results in the following manner: CGATCGA**[T/G]**GCTA

Example 1: pseudocode

```
1.  What do I have as inputs? Two sequences, which are
    the arguments of my function.
2.  We need an empty string to store the result
    sequence: create seqResult variable
3.  We need to compare the elements of the sequences at
    each position, therefore we have to iterate through
    all the elements by using a loop.
4.  We need to determine at the current position (index
    i) if these alleles are equal or not, therefore we
    perform two checks:
     a.  Check if both alleles are different
     b.  Check if both alleles are equal
5.  At each check we do different things on the growing
    sequence:
     a.  If they are different, append “variant
         string” at the current position (i) to
         seqResult.
     b.  If they are equal, append allele at the
         current position (i) to seqResult.
6.  Once finished the sequence, provide seqResult
    variable.
```

Example 1: translate pseudocode

1. What do I have as inputs? Two sequences, which are the arguments of my function.
2. We need an empty string to store the result sequence: create seqResult variable
3. We need to compare the elements of the sequences at each position, therefore we have to iterate through all the elements by using a loop.
4. We need to determine at the current position (index i) if these alleles are equal or not, therefore we perform two checks:
   a. Check if both alleles are different
   b. Check if both alleles are equal
5. At each check we do different things on the growing sequence:
   a. If they are different, append "variant string" at the current position (i) to seqResult.
   b. If they are equal, append allele at the current position (i) to seqResult.
6. Once finished the sequence, provide seqResult variable.

```
def variation(seq1, seq2):
    seqResult = ""
    for i in range(len(seq1)):
        if seq1[i] != seq2[i]:
            var = "["+seq1[i]+"/"+seq2[i]+"]"
            seqResult = seqResult + var
        else:
            seqResult = seqResult + seq1[i]
    return(seqResult)
```

Index												
Sequence 1	C	G	A	T	C	G	A	T	G	C	T	A
Sequence 2	C	G	A	T	C	G	A	G	G	C	T	A

Input objects

```
sequence1 = "CGATCGATGCTA"
sequence2 = "CGATCGAGGCTA"
```

Index												
Sequence 1	C	G	A	T	C	G	A	T	G	C	T	A
Sequence 2	C	G	A	T	C	G	A	G	G	C	T	A

Input objects

```
sequence1 = “CGATCGATGCTA”
sequence2 = “CGATCGAGGCTA”
```

Define function

```
def variation(seq1, seq2):
     seqResult = “ ”
     for i in range(len(seq1)):
          if seq1[i] != seq2[i]:
               var = “[“ + seq1[i] + “/” + seq2[i] + “]”
               seqResult = seqResult + var
          else:
               seqResult = seqResult + seq1[i]
     return(seqResult)
```

Call the function

```
result = variation(sequence1, sequence2)
```

Index												
Sequence 1	C	G	A	T	C	G	A	T	G	C	T	A
Sequence 2	C	G	A	T	C	G	A	G	G	C	T	A
seqResult												

Input objects

```
sequence1 = "CGATCGATGCTA"
sequence2 = "CGATCGAGGCTA"
```

Define function

```
def variation(seq1, seq2):
     seqResult = " "
     for i in range(len(seq1)):
          if seq1[i] != seq2[i]:
               var = "[" + seq1[i] + "/" + seq2[i] + "]"
               seqResult = seqResult + var
          else:
               seqResult = seqResult + seq1[i]
     return(seqResult)
```

Call the function

```
result = variation(sequence1, sequence2)
```

Index	0	1	2	3	4	5	6	7	8	9	10	11
Sequence 1	C	G	A	T	C	G	A	T	G	C	T	A
Sequence 2	C	G	A	T	C	G	A	G	G	C	T	A
seqResult												

Input objects
```
sequence1 = "CGATCGATGCTA"
sequence2 = "CGATCGAGGCTA"
```

Define function
```
def variation(seq1, seq2):
     seqResult = " "
     for i in range(len(seq1)):
          if seq1[i] != seq2[i]:
               var = "[" + seq1[i] + "/" + seq2[i] + "]"
               seqResult = seqResult + var
          else:
               seqResult = seqResult + seq1[i]
     return(seqResult)
```

Call the function
```
result = variation(sequence1, sequence2)
```

Index	0	1	2	3	4	5	6	7	8	9	10	11
Sequence 1	C	G	A	T	C	G	A	T	G	C	T	A
Sequence 2	C	G	A	T	C	G	A	G	G	C	T	A
seqResult												

?

Input objects

```
sequence1 = “CGATCGATGCTA”
sequence2 = “CGATCGAGGCTA”
```

Define function

```
def variation(seq1, seq2):
     seqResult = “ ”
     for i in range(len(seq1)):
          if seq1[i] != seq2[i]:
               var = “[“ + seq1[i] + “/” + seq2[i] + “]”
               seqResult = seqResult + var
          else:
               seqResult = seqResult + seq1[i]
     return(seqResult)
```

Call the function

```
result = variation(sequence1, sequence2)
```

Index	0	1	2	3	4	5	6	7	8	9	10	11
Sequence 1	C	G	A	T	C	G	A	T	G	C	T	A
Sequence 2	C	G	A	T	C	G	A	G	G	C	T	A
seqResult												

?

Input objects
```
sequence1 = “CGATCGATGCTA”
sequence2 = “CGATCGAGGCTA”
```

Define function
```
def variation(seq1, seq2):
     seqResult = “ ”
     for i in range(len(seq1)):
          if seq1[i] != seq2[i]:
               var = “[“ + seq1[i] + “/” + seq2[i] + “]”
               seqResult = seqResult + var
          else:
               seqResult = seqResult + seq1[i]
     return(seqResult)
```

TRUE!

Call the function
```
result = variation(sequence1, sequence2)
```

Index	0	1	2	3	4	5	6	7	8	9	10	11
Sequence 1	C	G	A	T	C	G	A	T	G	C	T	A
Sequence 2	C	G	A	T	C	G	A	G	G	C	T	A
seqResult												

?

Input objects

```
sequence1 = “CGATCGATGCTA”
sequence2 = “CGATCGAGGCTA”
```

Define function

```
def variation(seq1, seq2):
     seqResult = “ ”
     for i in range(len(seq1)):
          if seq1[i] != seq2[i]:
               var = “[“ + seq1[i] + “/” + seq2[i] + “]”
               seqResult = seqResult + var
          else:
               seqResult = seqResult + seq1[i]
     return(seqResult)
```

FALSE!

Call the function

```
result = variation(sequence1, sequence2)
```

Index	0	1	2	3	4	5	6	7	8	9	10	11
Sequence 1	C	G	A	T	C	G	A	T	G	C	T	A
Sequence 2	C	G	A	T	C	G	A	G	G	C	T	A
seqResult												

Input objects

```
sequence1 = “CGATCGATGCTA”
sequence2 = “CGATCGAGGCTA”
```

Define function

```
def variation(seq1, seq2):
     seqResult = “ ”
     for i in range(len(seq1)):
          if seq1[i] != seq2[i]:
               var = “[“ + seq1[i] + “/” + seq2[i] + “]”
               seqResult = seqResult + var
          else:
               seqResult = seqResult + seq1[i]
     return(seqResult)
```

Call the function

```
result = variation(sequence1, sequence2)
```

Index	0	1	2	3	4	5	6	7	8	9	10	11
Sequence 1	C	G	A	T	C	G	A	T	G	C	T	A
Sequence 2	C	G	A	T	C	G	A	G	G	C	T	A
seqResult												

Input objects

```
sequence1 = "CGATCGATGCTA"
sequence2 = "CGATCGAGGCTA"
```

Define function

```
def variation(seq1, seq2):
     seqResult = " "
     for i in range(len(seq1)):
          if seq1[i] != seq2[i]:
               var = "[" + seq1[i] + "/" + seq2[i] + "]"
               seqResult = seqResult + var
          else:
               seqResult = seqResult + seq1[i]
     return(seqResult)
```

Call the function

```
result = variation(sequence1, sequence2)
```

Environment

sequence1 = "CGAT..."
sequence2 = "CGAT..."
variation

Variables within the function

seqResult = " "

Index	0	1	2	3	4	5	6	7	8	9	10	11
Sequence 1	C	G	A	T	C	G	A	T	G	C	T	A
Sequence 2	C	G	A	T	C	G	A	G	G	C	T	A
seqResult												

Input objects
```
sequence1 = "CGATCGATGCTA"
sequence2 = "CGATCGAGGCTA"
```

Define function
```
def variation(seq1, seq2):
    seqResult = " "
    for i in range(len(seq1)):
        if seq1[i] != seq2[i]:
            var = "[" + seq1[i] + "/" + seq2[i] + "]"
            seqResult = seqResult + var
        else:
            seqResult = seqResult + seq1[i]
    return(seqResult)
```

(iteration 1)

Call the function
```
result = variation(sequence1, sequence2)
```

Environment

sequence1 = "CGAT..."
sequence2 = "CGAT..."
variation

Variables within the function

seqResult = " "
i = 0

Index	0	1	2	3	4	5	6	7	8	9	10	11
Sequence 1	C	G	A	T	C	G	A	T	G	C	T	A
Sequence 2	C	G	A	T	C	G	A	G	G	C	T	A
seqResult												

Input objects

```
sequence1 = "CGATCGATGCTA"
sequence2 = "CGATCGAGGCTA"
```

Define function

```
def variation(seq1, seq2):
    seqResult = " "
    for i in range(len(seq1)):
        if seq1[i] != seq2[i]:
            var = "[" + seq1[i] + "/" + seq2[i] + "]"
            seqResult = seqResult + var
        else:
            seqResult = seqResult + seq1[i]
    return(seqResult)
```

(iteration 1)

Call the function

```
result = variation(sequence1, sequence2)
```

Environment

sequence1 = “CGAT...”
sequence2 = “CGAT...”
variation

Variables within the function

seqResult = “ ”
i = 0

Index	0	1	2	3	4	5	6	7	8	9	10	11
Sequence 1	C	G	A	T	C	G	A	T	G	C	T	A
Sequence 2	C	G	A	T	C	G	A	G	G	C	T	A
seqResult	C											

Input objects
```
sequence1 = "CGATCGATGCTA"
sequence2 = "CGATCGAGGCTA"
```

Define function
```
def variation(seq1, seq2):
     seqResult = " "
     for i in range(len(seq1)):
          if seq1[i] != seq2[i]:
               var = "[" + seq1[i] + "/" + seq2[i] + "]"
               seqResult = seqResult + var
          else:
               seqResult = seqResult + seq1[i]
     return(seqResult)
```

(iteration 1)

Call the function
```
result = variation(sequence1, sequence2)
```

Environment

sequence1 = "CGAT..."
sequence2 = "CGAT..."
variation

Variables within the function

seqResult = "C"
i = 0

Index	0	1	2	3	4	5	6	7	8	9	10	11
Sequence 1	C	G	A	T	C	G	A	T	G	C	T	A
Sequence 2	C	G	A	T	C	G	A	G	G	C	T	A
seqResult	C	G										

Input objects
```
sequence1 = "CGATCGATGCTA"
sequence2 = "CGATCGAGGCTA"
```

Define function
```
def variation(seq1, seq2):
     seqResult = " "
     for i in range(len(seq1)):
          if seq1[i] != seq2[i]:
               var = "[" + seq1[i] + "/" + seq2[i] + "]"
               seqResult = seqResult + var
          else:
               seqResult = seqResult + seq1[i]
     return(seqResult)
```

(iteration 2)

Call the function
```
result = variation(sequence1, sequence2)
```

Environment

sequence1 = "CGAT..."
sequence2 = "CGAT..."
variation

Variables within the function

seqResult = "CG"
i = 1

Index	0	1	2	3	4	5	6	7	8	9	10	11
Sequence 1	C	G	A	T	C	G	A	T	G	C	T	A
Sequence 2	C	G	A	T	C	G	A	G	G	C	T	A
seqResult	C	G	A									

Input objects

```
sequence1 = "CGATCGATGCTA"
sequence2 = "CGATCGAGGCTA"
```

Define function

```
def variation(seq1, seq2):
     seqResult = " "
     for i in range(len(seq1)):
          if seq1[i] != seq2[i]:
               var = "[" + seq1[i] + "/" + seq2[i] + "]"
               seqResult = seqResult + var
          else:
               seqResult = seqResult + seq1[i]
     return(seqResult)
```

(iteration 3)

Call the function

```
result = variation(sequence1, sequence2)
```

Environment

sequence1 = “CGAT...”
sequence2 = “CGAT...”
variation

Variables within the function

seqResult = “CGA”
i = 2

Index	0	1	2	3	4	5	6	7	8	9	10	11
Sequence 1	C	G	A	T	C	G	A	T	G	C	T	A
Sequence 2	C	G	A	T	C	G	A	G	G	C	T	A
seqResult	C	G	A	T	C	G	A					

Input objects
```
sequence1 = "CGATCGATGCTA"
sequence2 = "CGATCGAGGCTA"
```

Define function
```
def variation(seq1, seq2):
    seqResult = " "
    for i in range(len(seq1)):
        if seq1[i] != seq2[i]:
            var = "[" + seq1[i] + "/" + seq2[i] + "]"
            seqResult = seqResult + var
        else:
            seqResult = seqResult + seq1[i]
    return(seqResult)
```

(iteration 8)

Call the function
```
result = variation(sequence1, sequence2)
```

Environment

sequence1 = "CGAT..."
sequence2 = "CGAT..."
variation

Variables within the function

seqResult = "CGATCGA"
i = 7

Index	0	1	2	3	4	5	6	7	8	9	10	11
Sequence 1	C	G	A	T	C	G	A	T	G	C	T	A
Sequence 2	C	G	A	T	C	G	A	G	G	C	T	A
seqResult	C	G	A	T	C	G	A	[T/G]				

Input objects
```
sequence1 = "CGATCGATGCTA"
sequence2 = "CGATCGAGGCTA"
```

Define function
```
def variation(seq1, seq2):
    seqResult = " "
    for i in range(len(seq1)):
        if seq1[i] != seq2[i]:
            var = "[" + seq1[i] + "/" + seq2[i] + "]"
            seqResult = seqResult + var
        else:
            seqResult = seqResult + seq1[i]
    return(seqResult)
```

(iteration 8)

Call the function
```
result = variation(sequence1, sequence2)
```

Environment

sequence1 = "CGAT..."
sequence2 = "CGAT..."
variation

Variables within the function

seqResult =
"CGATCGA[T/G]"
i = 7

End of the loop!

Index	0	1	2	3	4	5	6	7	8	9	10	11
Sequence 1	C	G	A	T	C	G	A	T	G	C	T	A
Sequence 2	C	G	A	T	C	G	A	G	G	C	T	A
seqResult	C	G	A	T	C	G	A	[T/G]	G	C	T	A

Input objects
```
sequence1 = "CGATCGATGCTA"
sequence2 = "CGATCGAGGCTA"
```

Define function
```
def variation(seq1, seq2):
     seqResult = " "
     for i in range(len(seq1)):
          if seq1[i] != seq2[i]:
               var = "[" + seq1[i] + "/" + seq2[i] + "]"
               seqResult = seqResult + var
          else:
               seqResult = seqResult + seq1[i]
     return(seqResult)
```

(iteration 12)

Call the function
```
result = variation(sequence1, sequence2)
```

Environment

sequence1 = "CGAT…"
sequence2 = "CGAT…"
variation

Variables within the function

seqResult =
"CGATCGA[T/G]GCTA"
i = 11

Index	0	1	2	3	4	5	6	7	8	9	10	11
Sequence 1	C	G	A	T	C	G	A	T	G	C	T	A
Sequence 2	C	G	A	T	C	G	A	G	G	C	T	A
seqResult	C	G	A	T	C	G	A	[T/G]	G	C	T	A

Input objects

```
sequence1 = “CGATCGATGCTA”
sequence2 = “CGATCGAGGCTA”
```

Define function

```
def variation(seq1, seq2):
     seqResult = “ ”
     for i in range(len(seq1)):
          if seq1[i] != seq2[i]:
               var = “[“ + seq1[i] + “/” + seq2[i] + “]”
               seqResult = seqResult + var
          else:
               seqResult = seqResult + seq1[i]
     return(seqResult)
```

Call the function

```
result = variation(sequence1, sequence2)
```

Environment

sequence1 = “CGAT...”
sequence2 = “CGAT...”
variation
result

Variables within the function

seqResult =
“CGATCGA[T/G]GCTA”
i = 11

Example 2

Exercise: Write a function that counts the number of occurrences of a certain subsequence (e. "ATG") in a longer sequence.

Sequence 1: CGATCGATGCTA

Example 2: pseudocode

1. What do I have as inputs? One sequence and one subsequence (arguments of my function).
2. Our goal is to count occurrences, which means that our result will be an integer. This is a growing number, therefore we need to initialize an integer variable.
3. We go through the sequence (iterable) in order to see all the possible subsequences. We use a loop.
4. At each position (i) of the sequence we obtain a chunk of length equal to the subsequence.
5. At each chunk that we extract from the sequence, we test if is equal to the subsequence we are testing.
    a. If the chunk is equal to the subsequence, our result variable (integer) grows by 1.
    b. If not, we continue looking for occurrences.

Index	0	1	2	3	4	5	6	7	8	9	10	11
Sequence	C	G	A	T	C	G	A	T	G	C	T	A

Subsequence: **ATG**

Example 3

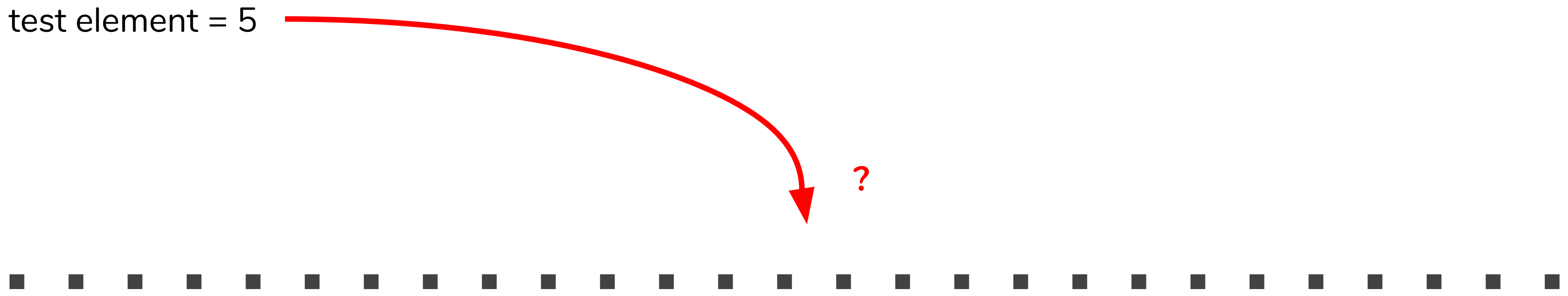
Exercise: Given a sorted list of numbers *test_list*, try to find the index of a certain number *test_number* within that list. Here we are going to make use of a binary search algorithm (methodology of **divide and conquer**).

test_list: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]

test_number: 8

Divide and conquer?

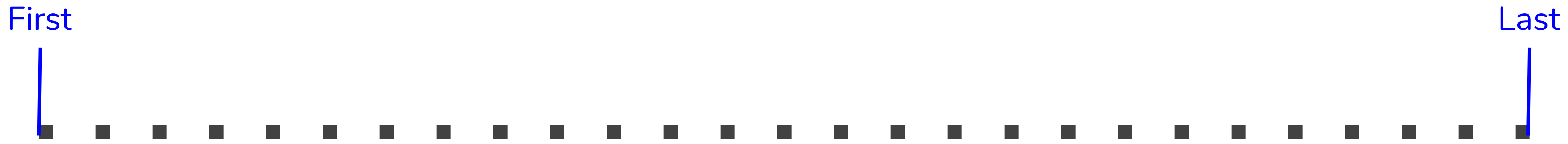
1. Break down a problem into two or more subproblems
2. Subproblems are easier to solve
3. Combination of subproblem results solve the global problem



Divide and conquer?

1. Break down a problem into two or more subproblems
2. Subproblems are easier to solve
3. Combination of subproblem results solve the global problem

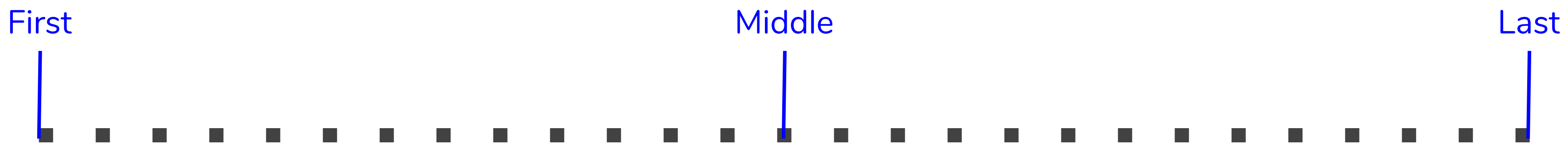
test element = 5



Divide and conquer?

1. Break down a problem into two or more subproblems
2. Subproblems are easier to solve
3. Combination of subproblem results solve the global problem

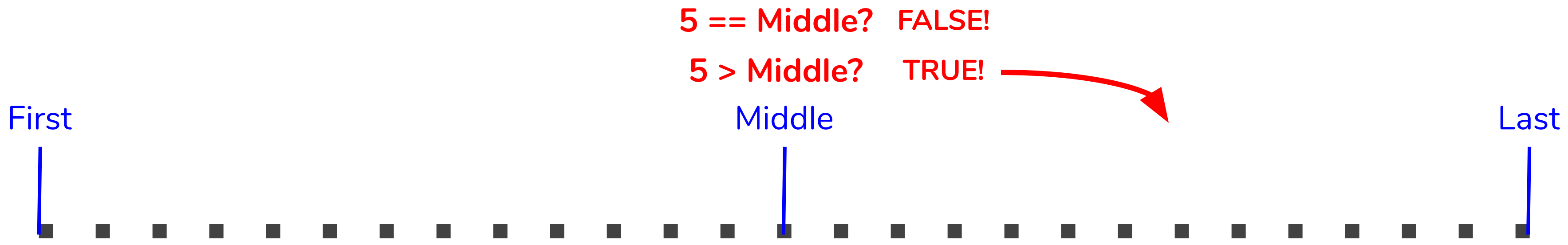
test element = 5



Divide and conquer?

1. Break down a problem into two or more subproblems
2. Subproblems are easier to solve
3. Combination of subproblem results solve the global problem

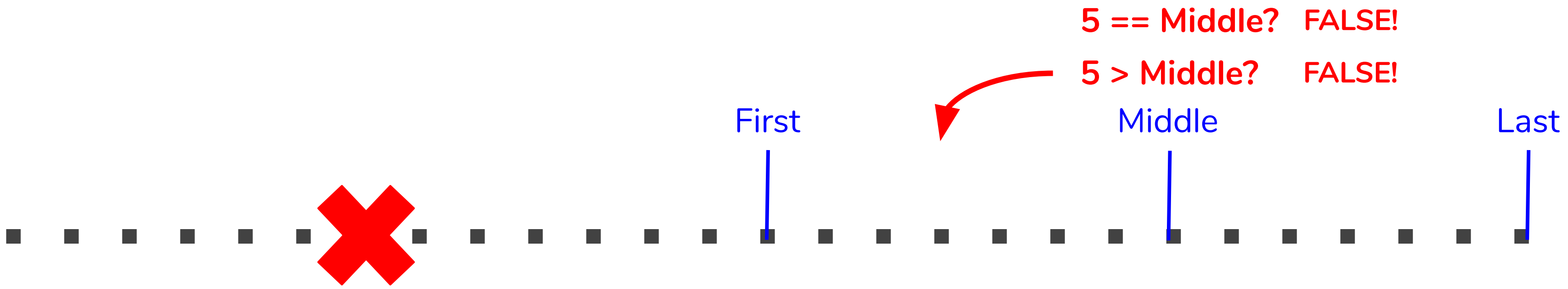
test element = 5



Divide and conquer?

1. Break down a problem into two or more subproblems
2. Subproblems are easier to solve
3. Combination of subproblem results solve the global problem

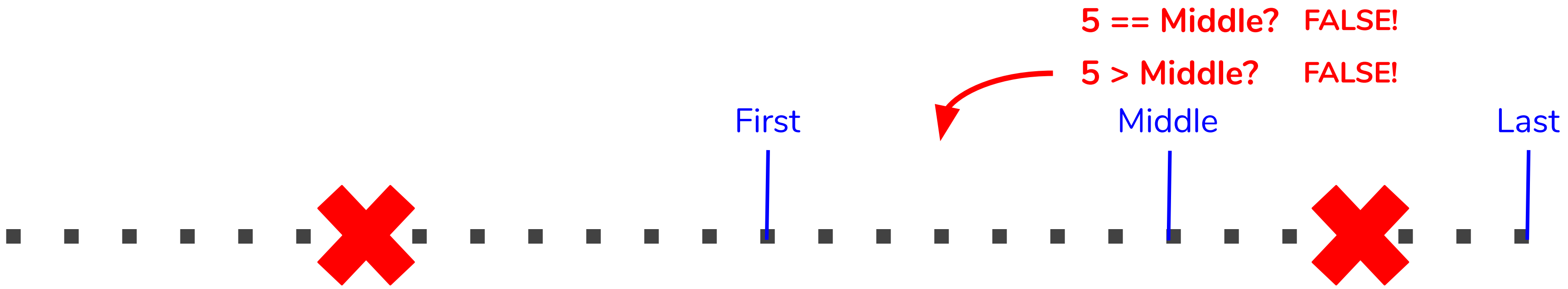
test element = 5



Divide and conquer?

1. Break down a problem into two or more subproblems
2. Subproblems are easier to solve
3. Combination of subproblem results solve the global problem

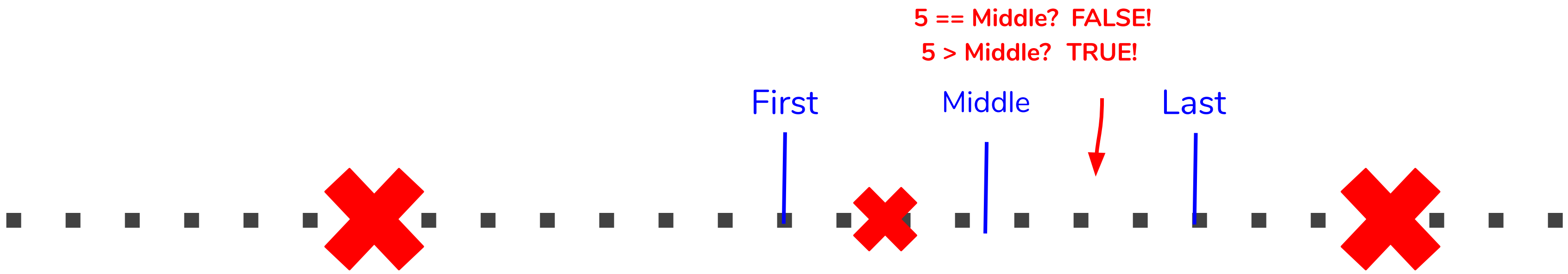
test element = 5



Divide and conquer?

1. Break down a problem into two or more subproblems
2. Subproblems are easier to solve
3. Combination of subproblem results solve the global problem

test element = 5



Example 3: pseudocode

1. What do I have as inputs? One numeric list and the number I want to find
2. Since the beginning we need to use **three variables** in order to **compare our number with the middle number**: the index of the first element of the list, the last index and the index of the number we want to find. **Why do we need this last one?**
3. Each time we are choosing between the upper/lower half of the list where to look for our number. However, the first and the last indexes of these lists must be **always** like first <= last. **What happens when we finally reach a list of two numbers? What is the middle number of this list?**
4. We need to find the element in the **middle** of the list. We use the indexes to find it (first+last/2).
5. We have to check if our number is equal to the number in the middle. In this check we have two possibilities:
   a. Our number is equal to the middle number. **We can return the index!**
   b. Our number is different to the middle number. Then we have to **update** (change) the extreme indexes (first or last) in order to take the upper or lower half list. How?
      i. If our number is lower than the middle number, the last index is equal to the middle index **minus** 1
      ii. Else (our number is higher than the middle number), the first index is equal to the middle index **plus** 1

Example 3: pseudocode

1. What do I have as inputs? One numeric list and the number I want to find
2. Since the beginning we need to use **three variables** in order to **compare our number with the middle number**: the index of the first element of the list, the last index and the index of the number we want to find. **Why do we need this last one?**
3. Each time we are choosing between the upper/lower half of the list where to look for our number. However, the first and the last indexes of these lists must be **always** like first <= last. **What happens when we finally reach a list of two numbers? What is the middle number of this list?**

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
List	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20