Linear Systems and Applications

A Hands-On Python Workshop

Rhudaina Mohammad

Institute of Mathematics, UP Diliman

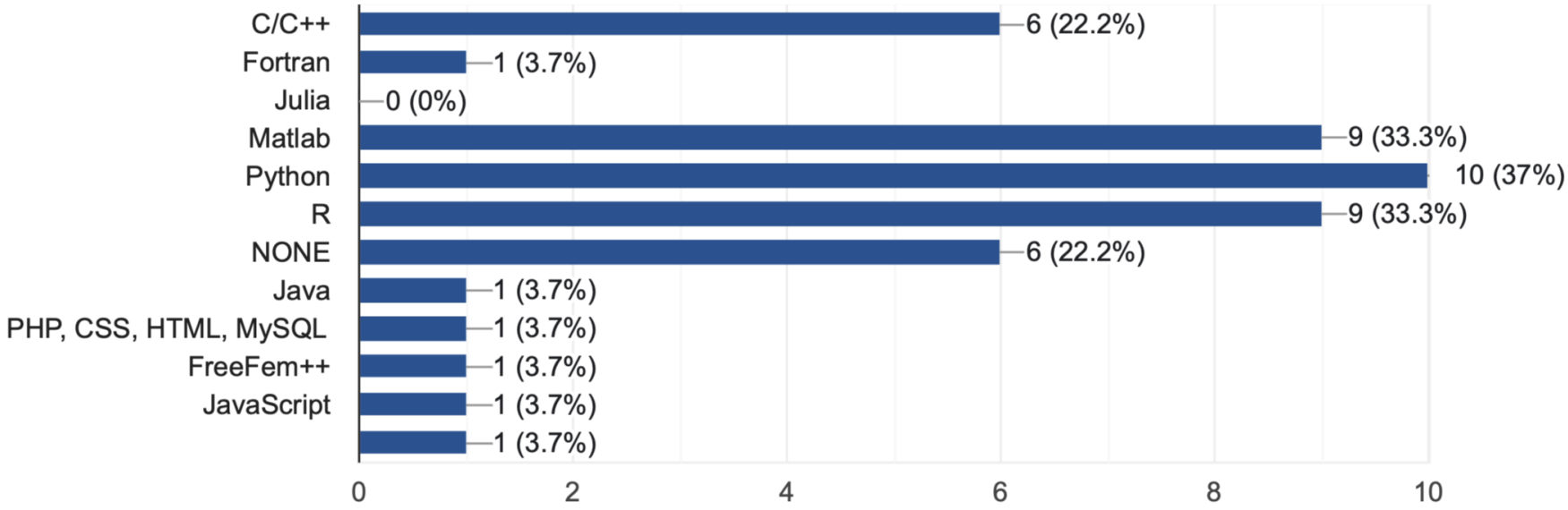
rzmohammad@up.edu.ph

23-25 May 2024 · CSRC Building
National Science Complex, UP Diliman

Outline

Day 1	Python Basics and Programming Fundamentals.
Day 2	Introduction to Numerical Methods for Linear Systems and Applications.
Day 3	Introduction to Dimensionality Reduction in Data Science.

Programming Languages



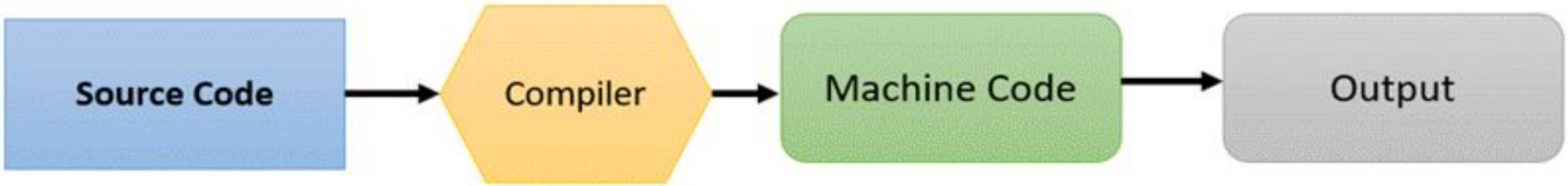
- There are two types:
  - minimize computing time (e.g. C/C++, Fortran)

    compiled language
  - minimize human coding time (e.g. Python, R, MATLAB)

    interpreted language

Compiled vs Interpreted

- Compiled Languages



- Interpreted Languages

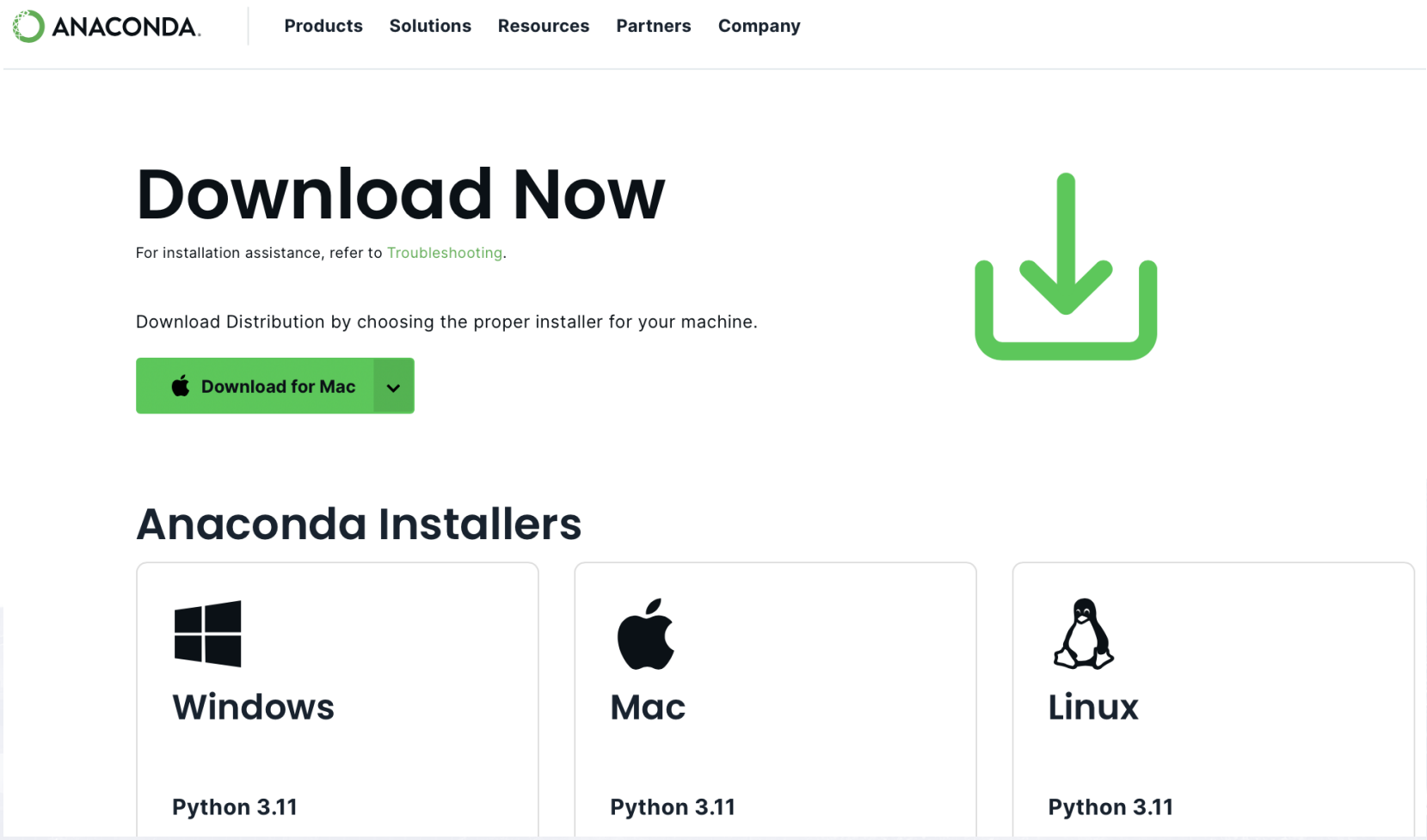
What is Python?

- high-level interpreted language : source code of a Python program is converted into bytecode that is then executed by the Python virtual machine
- dynamically-typed : variable types are determined and checked at runtime rather than during compilation
- garbage collector : automatic memory management
- created by Guido van Rossum and first released on 20 February 1991
- named after the 1970s BBC comedy series ”Monty Python’s Flying Circus”

Work offline...

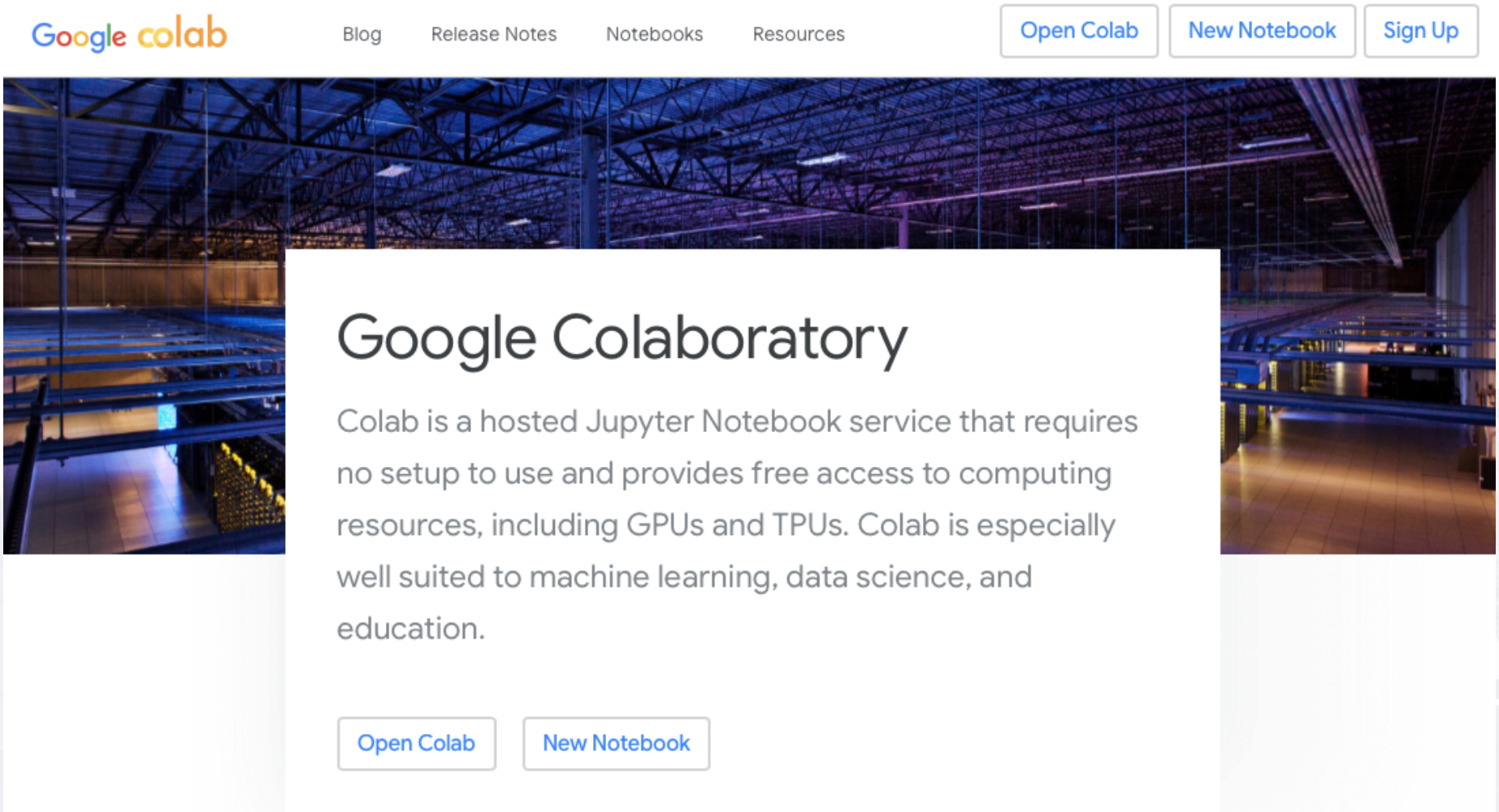
- Install anaconda (with jupyter support)

`https://www.anaconda.com/download`



Work online...

- Google Colab (`https://colab.google.com`)
  - a hosted Jupyter Notebook service that requires no setup to use and provides free access to computing resources
  - suited for machine learning, data science, and education



Outline

https://github.com/rhudaina/Linear-Systems-and-Applications-A-Hands-On-Python-Workshop

Built-in Functions

- objects that can be used by all Python code without the need of an `import` statement

`abs()`	`input()`	`min()`	`round()`
`dir()`	`int()`	`open()`	`str()`
`divmod()`	`len()`	`pow()`	`sum()`
`float()`	`list()`	`print()`	`type()`
`help()`	`max()`	`range()`	

math package

```
1 import math
```

- provides access to the mathematical functions defined by the C standard
- Number-theoretic and representation functions:

```
math.ceil()     math.fabs()     math.factorial()
math.floor()    math.fsum()     math.gcd()
```

- Power and logarithmic functions:

```
math.exp()    math.pow()     math.sqrt()
math.log()    math.log2()    math.log10()
```

- Trigonometric functions:

```
math.cos()     math.sin()     math.tan()
math.acos()    math.asin()    math.atan()
```

math package

```
1 import math
```

- Angular conversion:
  `math.degrees()` `math.radians()`
- Hyperbolic functions:
  `math.cosh()` `math.sinh()` `math.tanh()`
  `math.acosh()` `math.asinh()` `math.atanh()`
- Constants:
  `math.pi()` `math.e()` `math.tau()`

```
1 from math import pi,sin,cos
2 sin(3*pi/4)
3 f = lambda x: sin(x)*cos(x)
```

numpy package

```
1 import numpy as np
```

- `numpy` (Numerical Python), a foundational library for scientific computing in

	`np.sqrt()`	`np.sin()`	`np.exp()`
Python	`np.zeros()`	`np.ones()`	`np.array()`
	`np.arange()`	`np.linspace()`	`np.meshgrid()`

- `np.zeros` command builds an array of zeros, often used for pre-allocating memory

```
1 A = np.zeros((3,5)) # 3x5 matrix of zeros
2 print(A)
```

- `np.ones` command builds an array of ones

```
1 B = np.ones((3,5)) # 3x5 matrix of ones
2 print(B)
```

numpy package

- `np.array` command builds an array of floating point numbers
  - memory usage is more efficient

```
1 M = np.array([[1,2,3],[4,5,6],[7,8,9]]) # matrix
2 u = np.array([[1],[2],[3]]) # column vector
3 v = np.array([1,2,3]) # row vector
```

- reading individual elements from a `numpy` array is the same as reading elements from a Python list

```
1 print(M[0,0]) # top left
2 print(M[2,2]) # bottom right
3 print(M[0,:]) # first row
4 print(M[:,2]) # second row
```

  - *Note:* Reading a column from a matrix will automatically flatten it into an array, not a column matrix

numpy package

- `np.array` command builds an array of floating point numbers
  - memory usage is more efficient

```
1 M = np.array([[1,2,3],[4,5,6],[7,8,9]]) # matrix
2 u = np.array([[1],[2],[3]]) # column vector
3 v = np.array([1,2,3]) # row vector
```

- there are built-in functions that can act directly on the `numpy` array as a matrix or a vector
  - `variable.shape` command gives the shape `(rows, columns)` of a `numpy` array

```
1 print(M.shape)
2 print(u.shape)
```

  - `variable.size` command gives the size (total number number of elemennts) of a `numpy` array

```
1 print(M.size)
2 print(v.size)
```

numpy package

- In Python, the default notion of multiplication is NOT matrix multiplication, but element-wise multiplication

```
1 A = np.array([[1,2],[3,4]])
2 w = np.array([[5],[6]])
3 print(A * A) # element-wise multiplication
4 print(A * w) # element-wise multiplication on
    each column
```

- Recasting `numpy` arrays as matrices allows matrix multiplication

```
1 A = np.matrix([[1,2],[3,4]])
2 w = np.matrix([[5],[6]])
3 print(A * A) # matrix multiplication
4 print(A * w)
5 print(A.T) # matrix transpose
6 print(A.I) # matrix inverse
```

numpy package

- `arange(start, stop, step)` builds an array of floating point numbers from `start` (default: 0) up to, but not including, `stop` with increment `step` (default: 1)

```
1 x = np.arange(0,5,0.5)
2 print(x)
```

- `linspace(start, stop, number of points)` builds an array of floating point numbers with exactly the `number of points` specified from `start` to `stop` with equal spacing in between

```
1 y = np.linspace(0,5,10)
2 print(y)
```

- `meshgrid()` creates a mesh grid

```
1 X, Y = np.meshgrid(x, y)
2 print("X = ", X)
3 print("Y = ", Y)
```

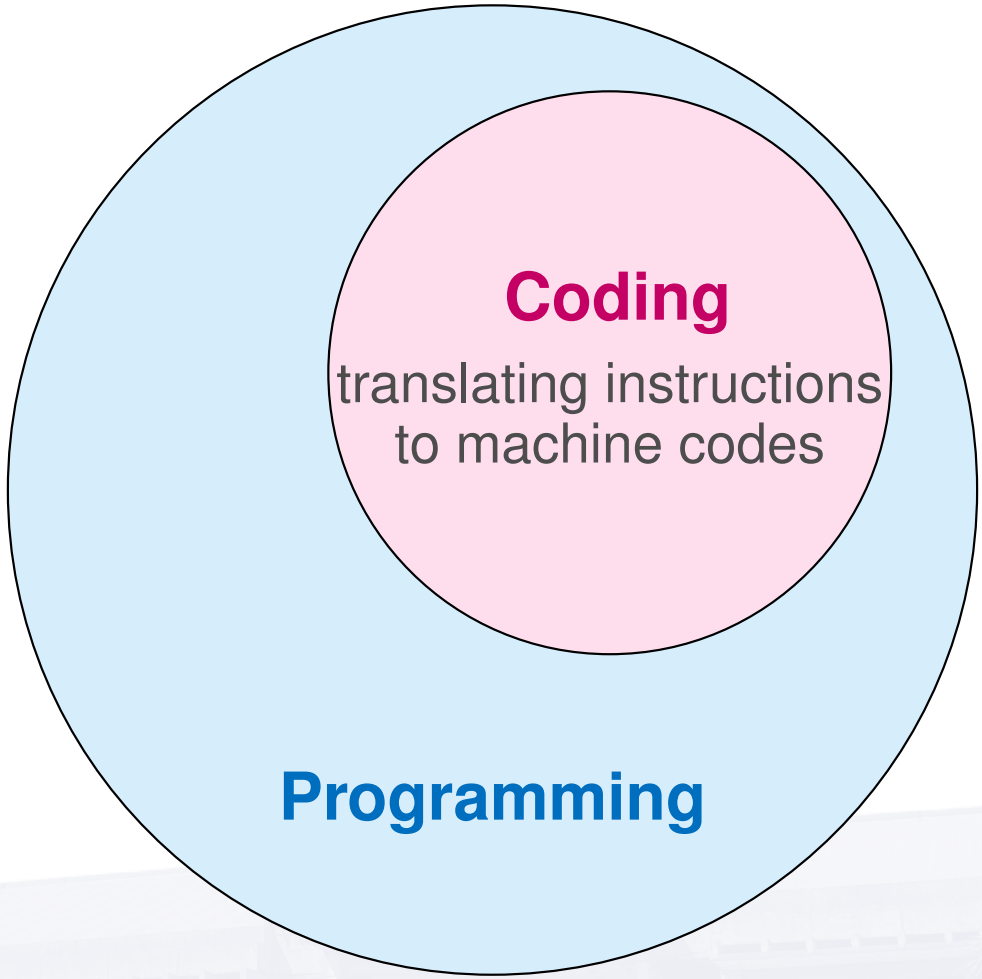
matplotlib package

```
1 import matplotlib.pyplot as plt
```

- `matplotlib.pyplot`, a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms

```
1  x = np.linspace(0,2*np.pi, 100)
2  y = np.sin(x)
3  z = np.cos(x)
4  plt.plot(x, y, 'b-', label='$\sin(x)$')
5  plt.plot(x, z, 'r--', label='$\cos(x)$')
6  plt.xlabel("$x$")
7  plt.ylabel("$y$")
8  plt.title("Sine and Cosine")
9  plt.legend()
10 plt.show()
```

Programming vs Coding



- creating and developing an executable machine program
- debugging and testing
- documentation review and analysis

Algorithm

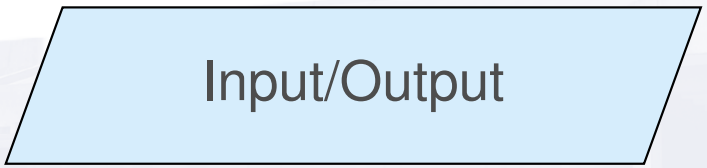
Algorithm is a procedure that describes, in unambiguous manner, a finite sequence of steps to be performed in a specified order

- presented in natural language, NOT programming language

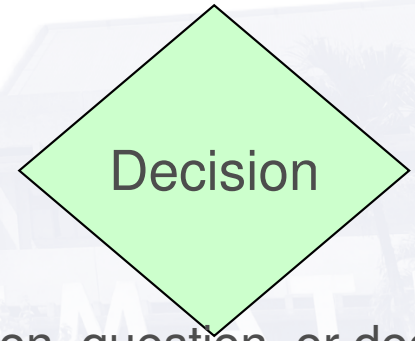
- To describe an algorithm:
  - pseudocode uses code-like statements
  - flowchart is a visual or graphical representation

beginning or end of the algorithm

calculations or data manipulations



inputs or outputs of algorithm



comparison, question, or decision that determines alternative paths to be followed

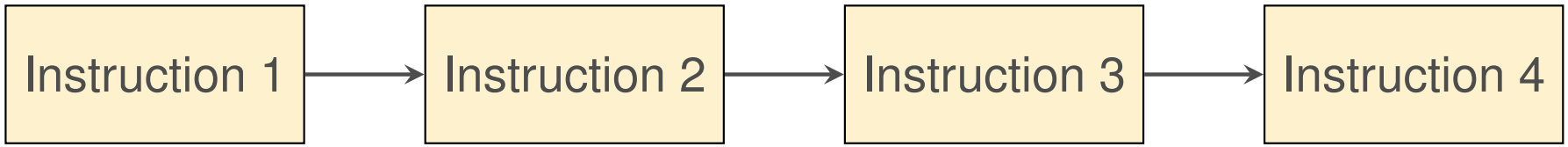
Use: arrows to represent the flow of logic

Control Structure

Control structures change the order that statements are executed or decide if a certain statement will be run

Fundamental control structures:

- Sequence expresses the trivial idea that unless you direct it otherwise, the computer code is to be implemented one instruction at a time



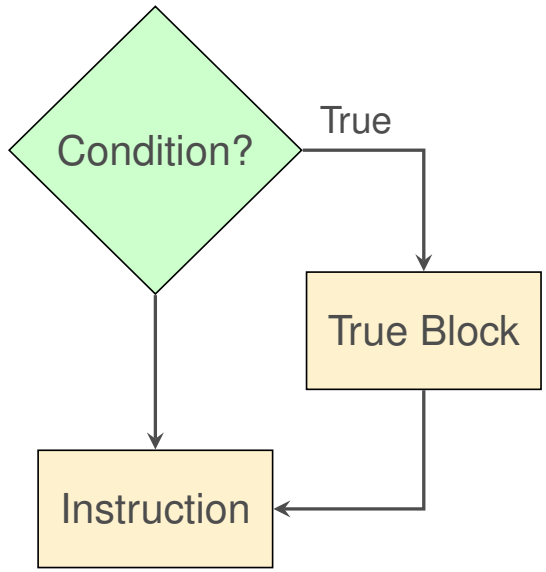
- Selection provides a means to split the flow into branches based on the outcome of a logical condition
- Repetition provides a means to implement instructions repeatedly

Selection

Single-alternative decision

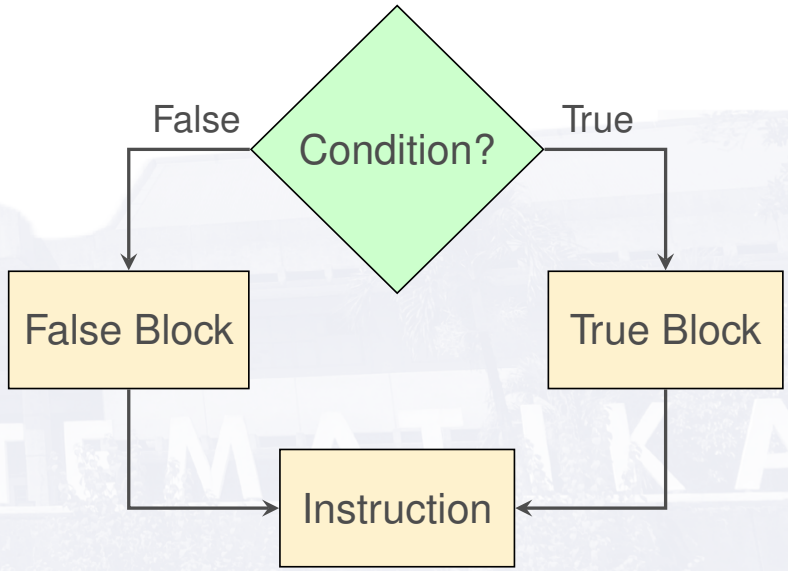
```
1 if condition :
2     TRUE block
```

```
1 if condition1 :
2     Block1
3     if condition2 :
4         Block2
```



Double-alternative decision

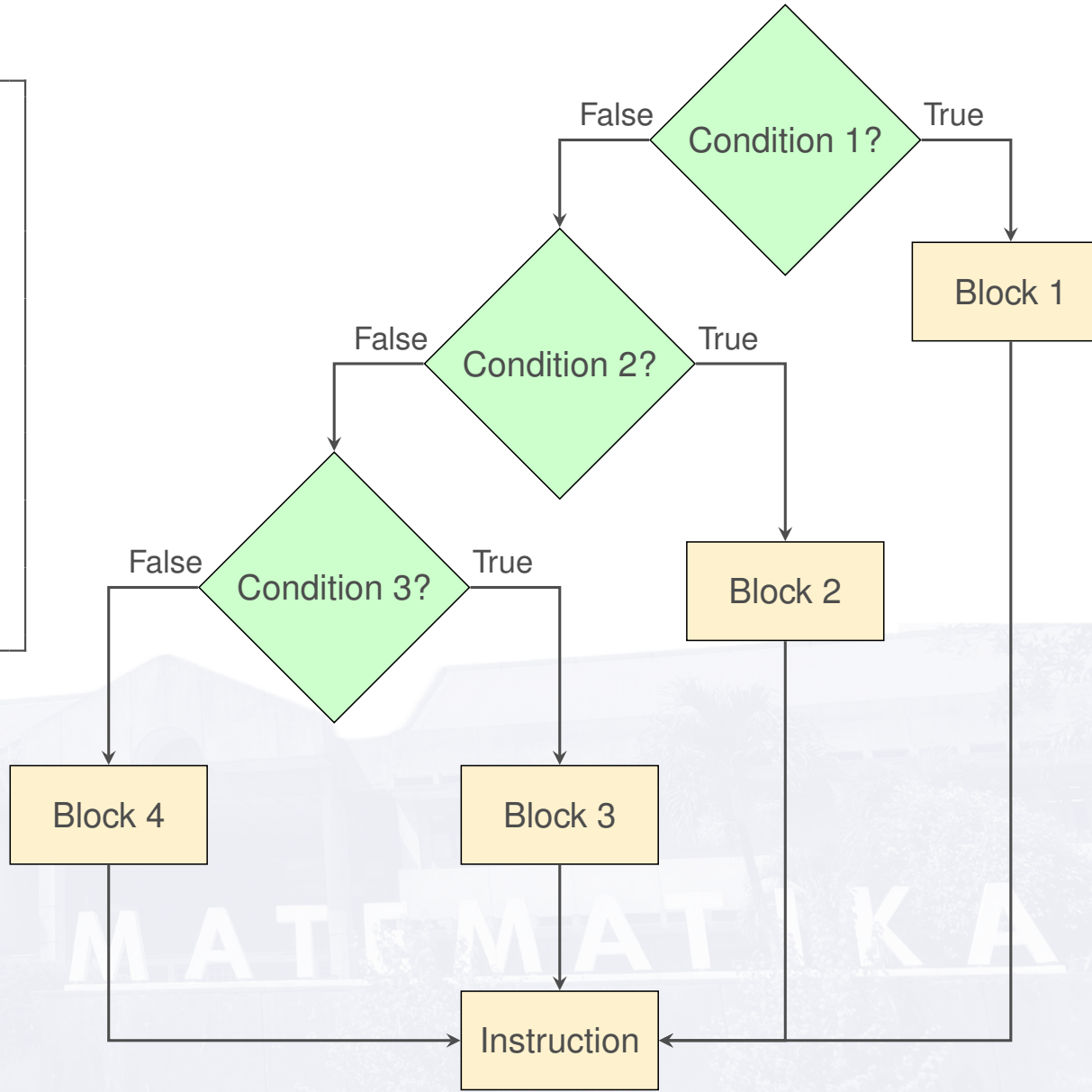
```
1 if condition :
2     TRUE block
3 else:
4     FALSE block
```



Selection

- Multialternative decision

```
1 if condition1 :
2     Block 1
3 elif condition2 :
4     Block 2
5 elif condition3 :
6     Block 3
7 else:
8     Block 4
```



Repetition

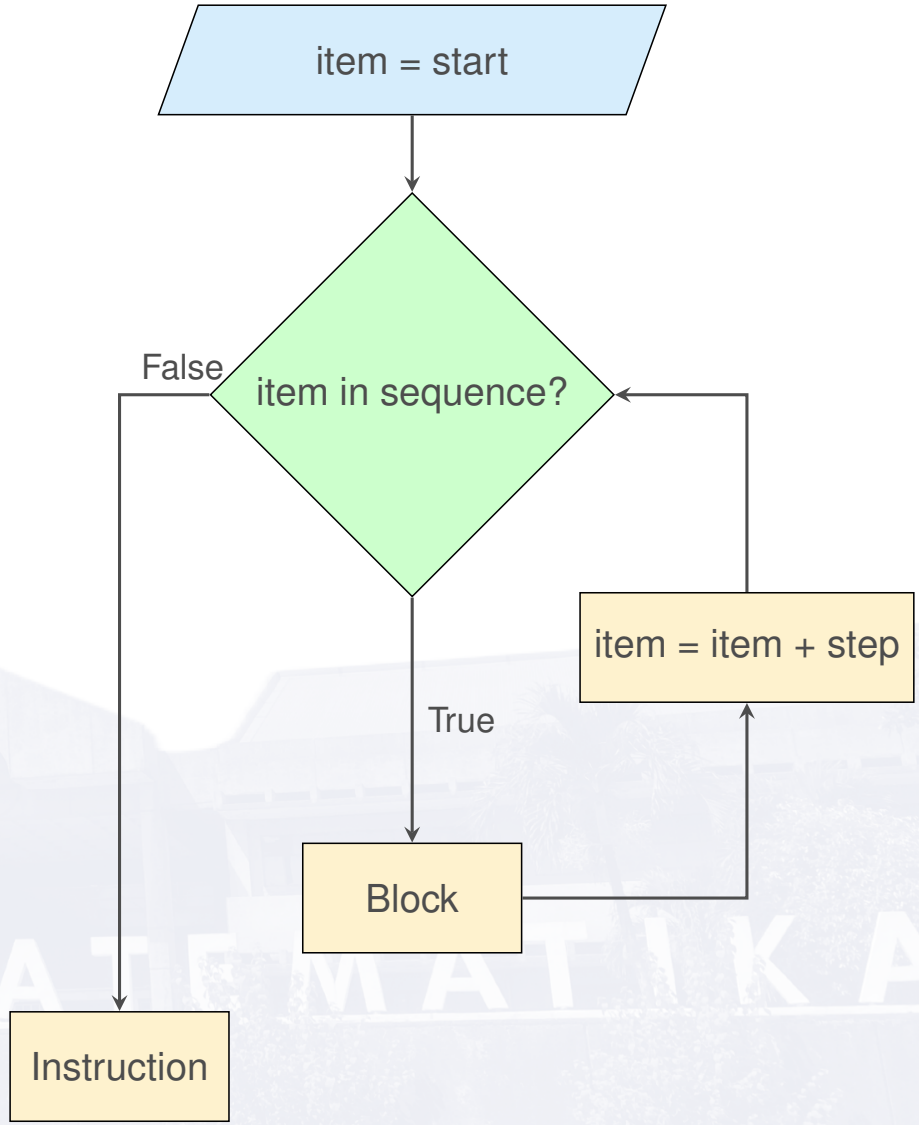
- Count-controlled loop performs a specified number of repetitions or iterations

```
1 for item in sequence:
2     Block
```

```
1 for i in range(8):
2     print(i)
```

```
1 for i in range(3,8):
2     print(i)
```

```
1 for i in
    range(3,10,2):
2     print(i)
```

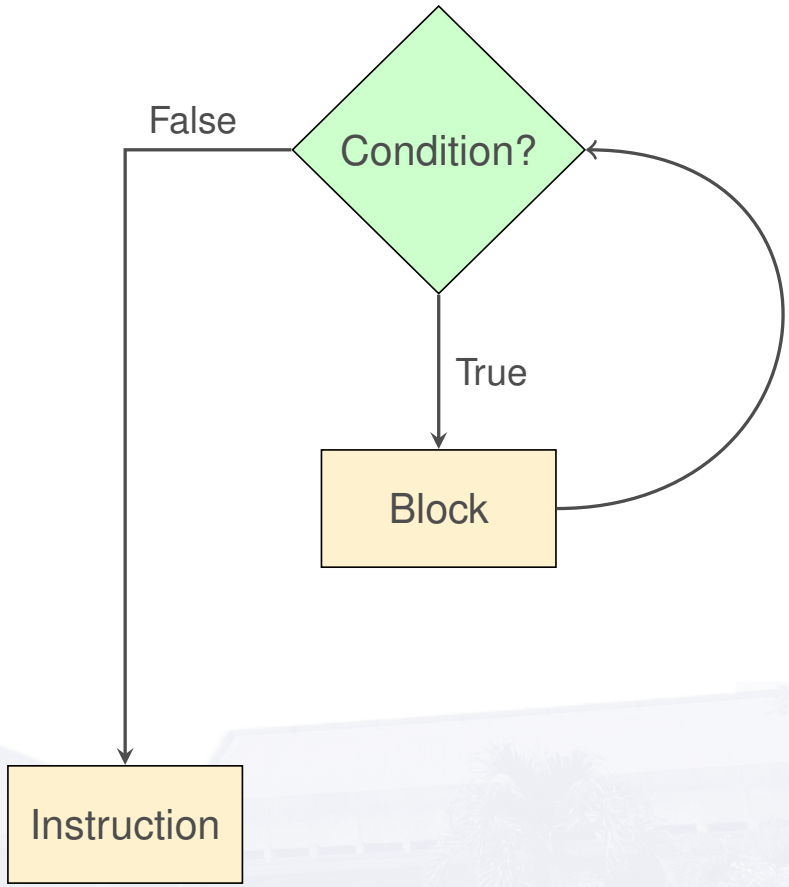


Repetition

- Decision loop terminates based on the result of a logical condition

```
1 while condition :
2     Block
```

```
1 count = 0
2 while count < 10 :
3     count += 1
4     print(count)
5 print("End")
```

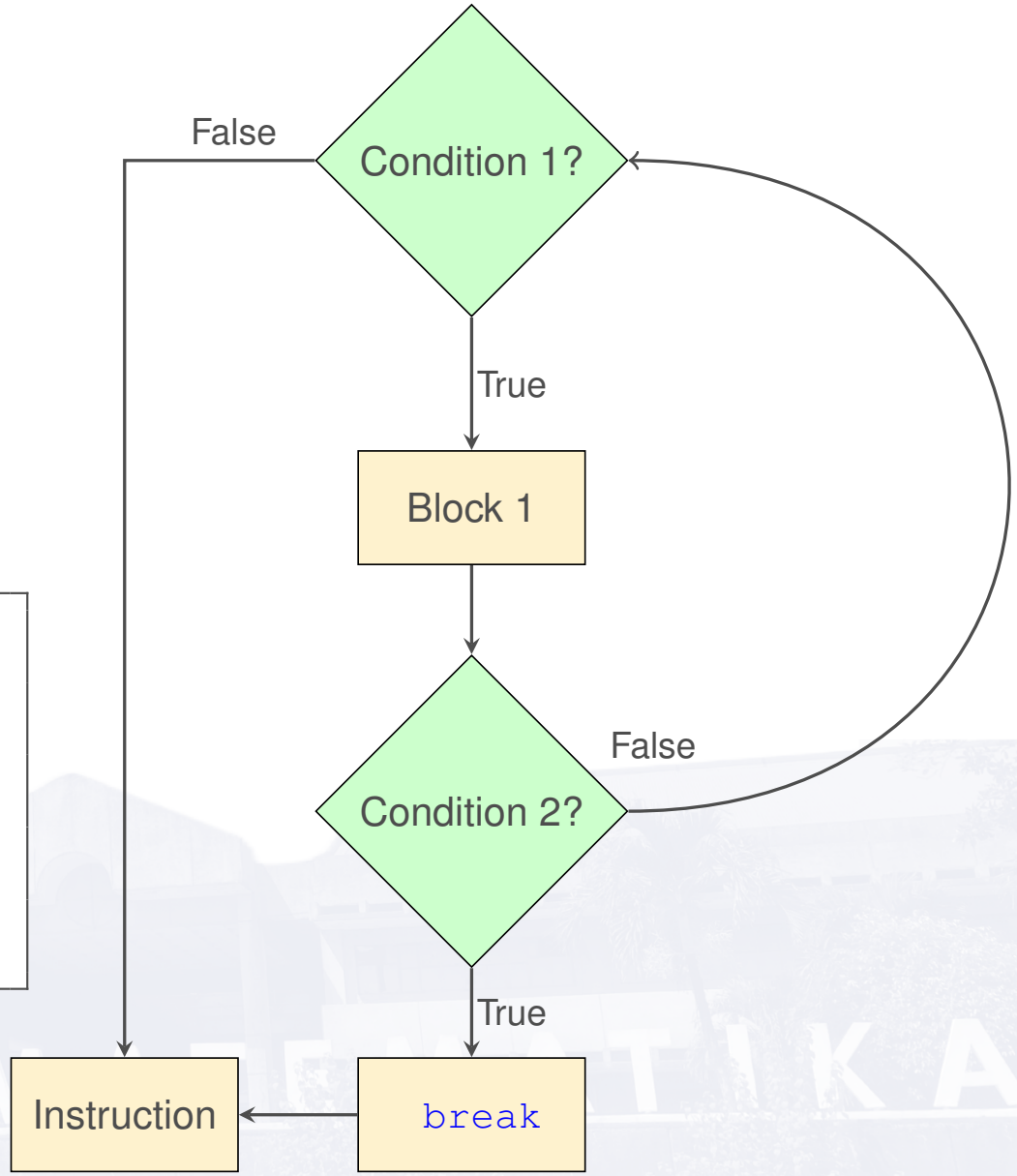



```
1 while 1 == 1:
2    print("Help, I'm stuck in a loop.")
```

break

- ”breaks out” of the loop
- terminates the loop immediately when it is encountered
- used with decision making statement, such as `if`

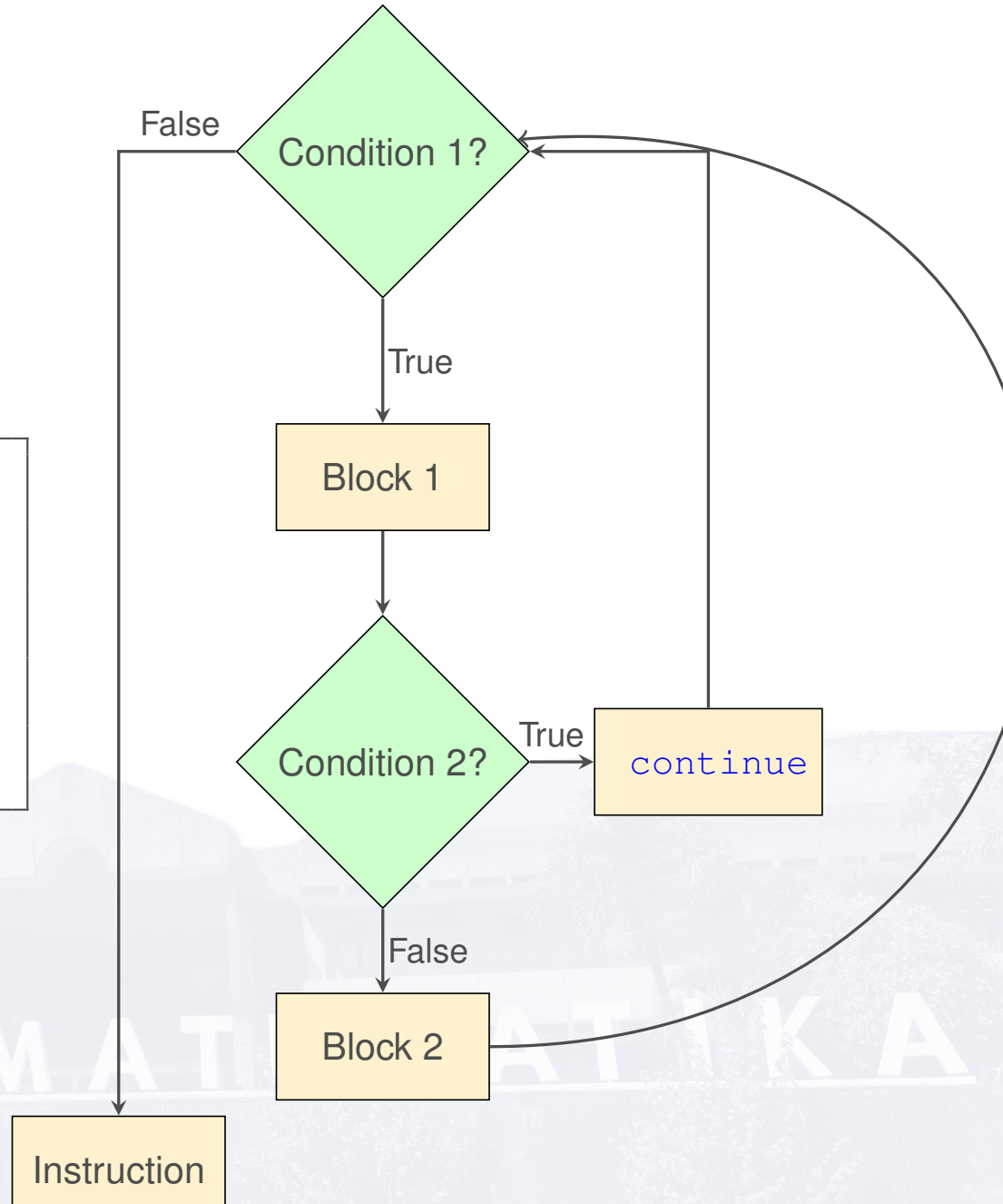
```
1 while condition1 :
2     Block 1
3     if condition2 :
4         break
5 Block 2
```



continue

- skips some statements inside the loop
- used with decision-making statement, such as `if`

```
1 while condition1 :
2     Block 1
3     if condition2 :
4         continue
5     Block 2
```



break vs continue

break

```
1 var = 10
2
3 while var > 0:
4     var -= 1
5     if var == 5:
6         break
7     print(var)
8
9 print("Goodbye!")
```

continue

```
1 var = 10
2
3 while var > 0:
4     var -= 1
5     if var == 5:
6         continue
7     print(var)
8
9 print("Goodbye!")
```

Thank you for your attention!