ES 691
Mathematics for Machine Learning

with

Dr. Naveed R. Butt

@

GIKI - FES

In the last two lectures we saw…

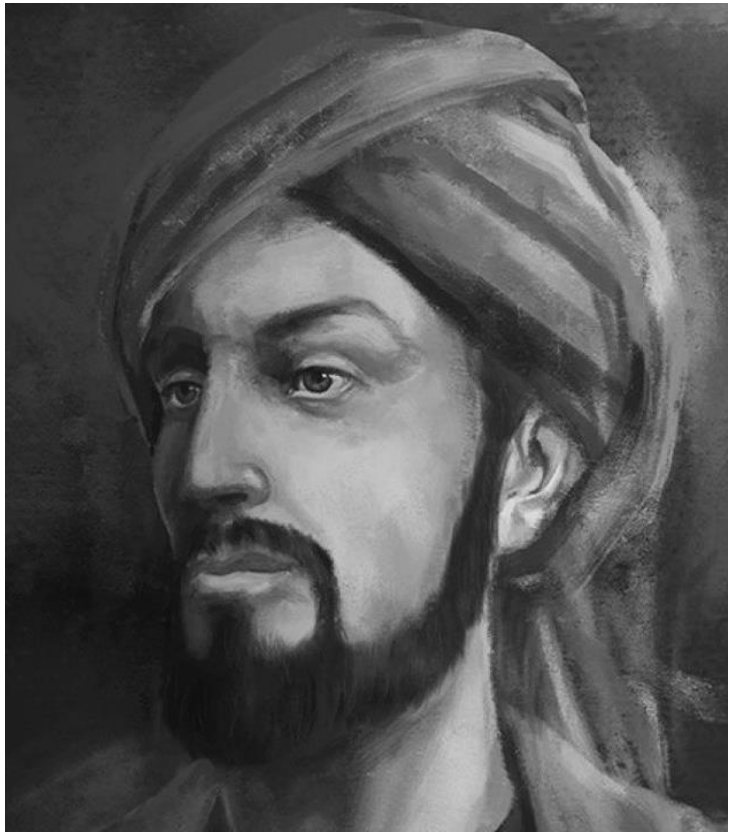
Vocabulary of Machine Learning

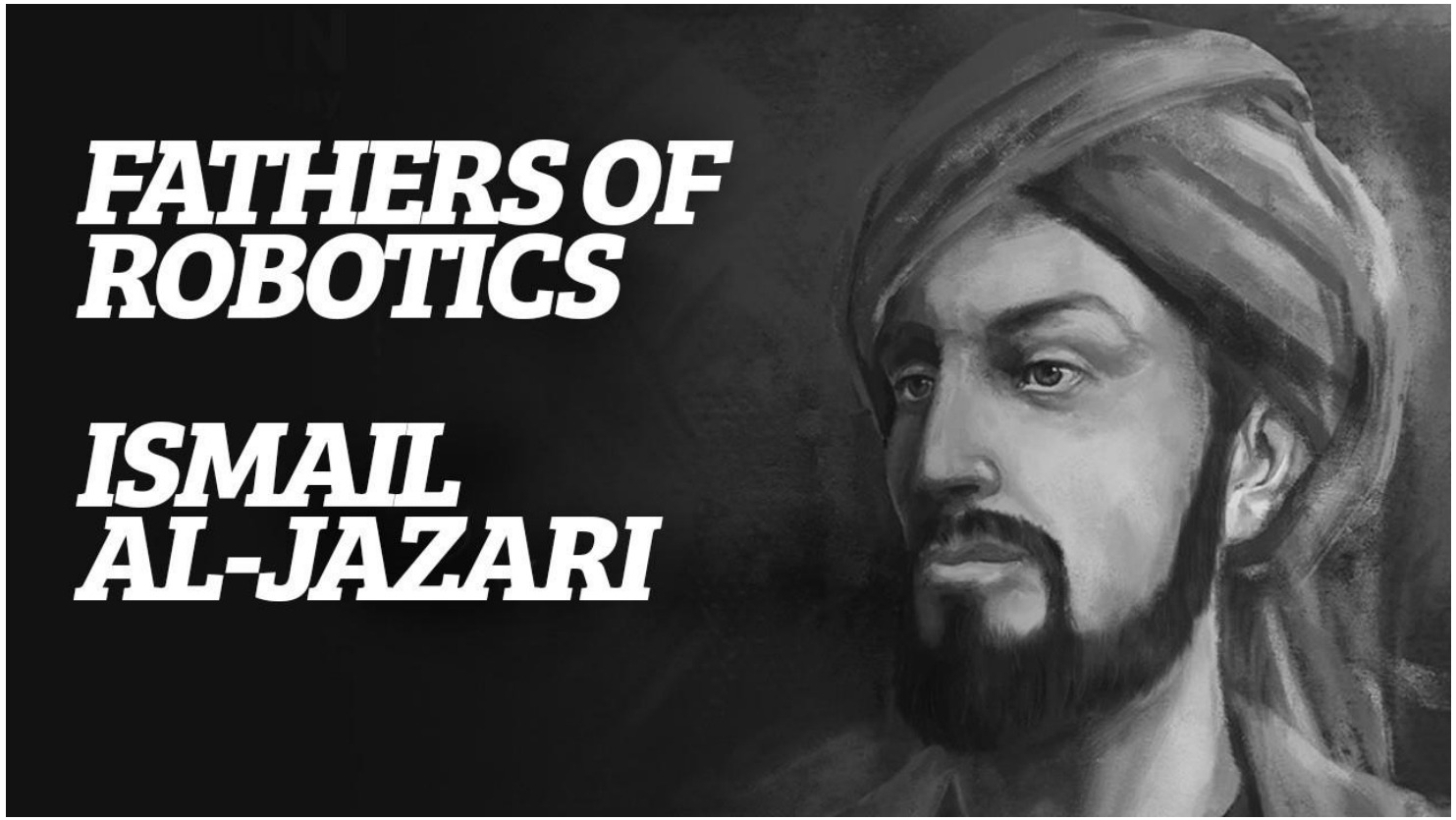
- General Terms
- Broad Classes – AI, ML, Deep ML, Generative ML
- ML Map and Associated Vocabulary
- Neural Networks Map and Associated Vocabulary

Next…

- A bit of history
- Python

The Rise and Fall
(and Rise and Fall)
and Rise of Machine Learning




FATHERS OF
ROBOTICS
ISMAIL
AL-JAZARI

The Book of Knowledge of Ingenious Mechanical Devices

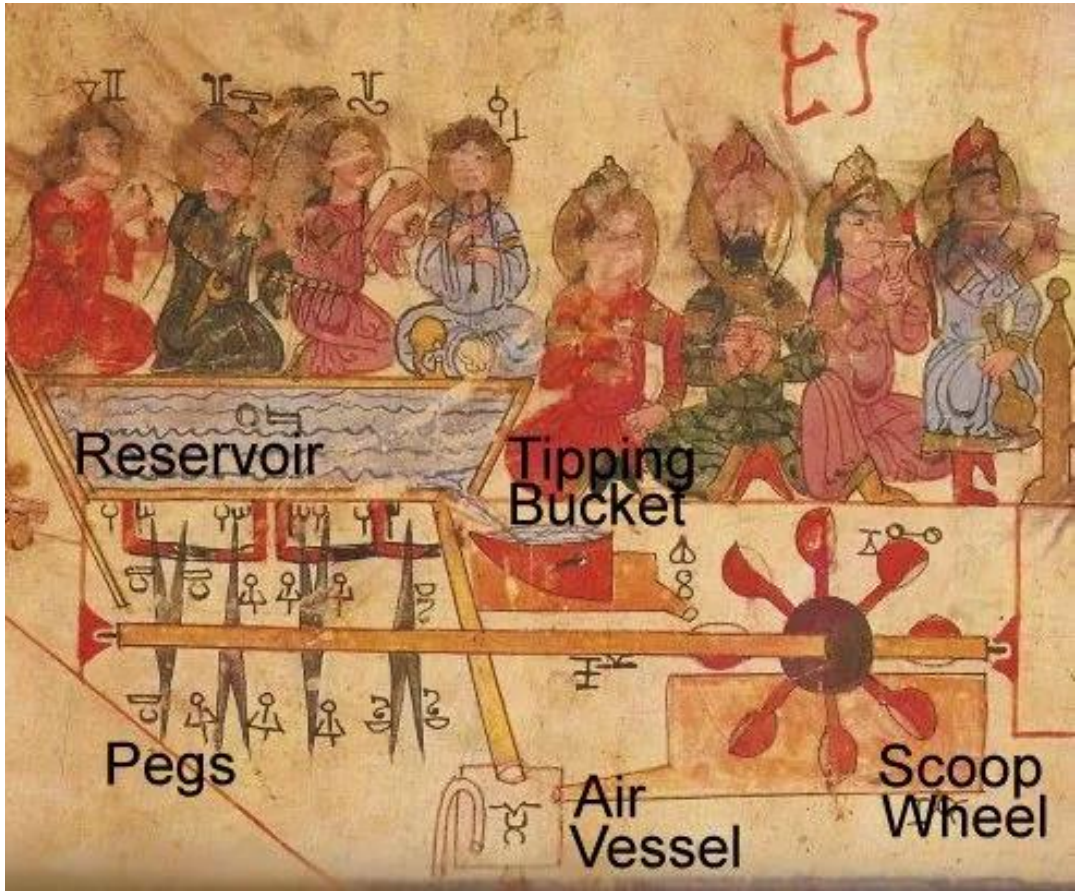
(Kitāb fī ma ʿrifat al-ḥiyal al-handasiyya)

by

Ibn al-Razzāz al-Jazarī

Al-Jazarī Creates the First Recorded Designs of a Programmable Automaton

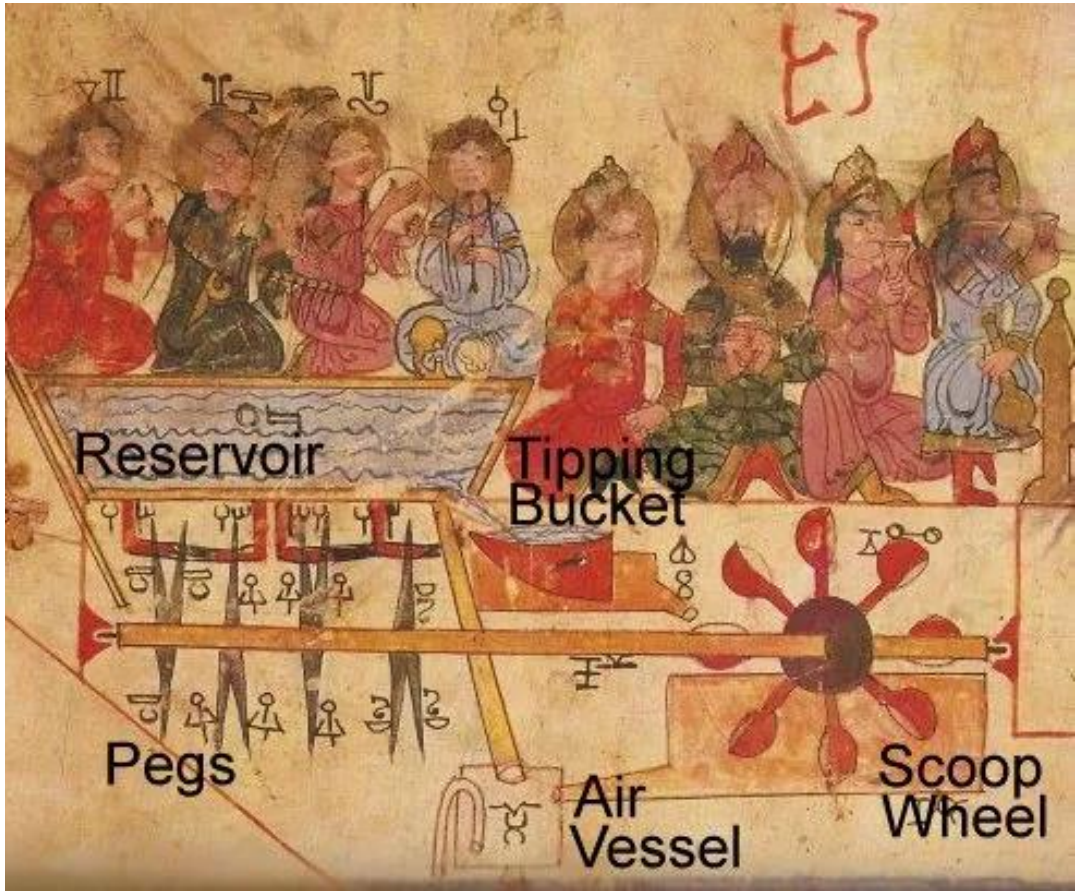
1206



Al-Jazarī Creates the First Recorded Designs of a Programmable Automaton

1206

Automaton [Latin] : "acting of one's own will"



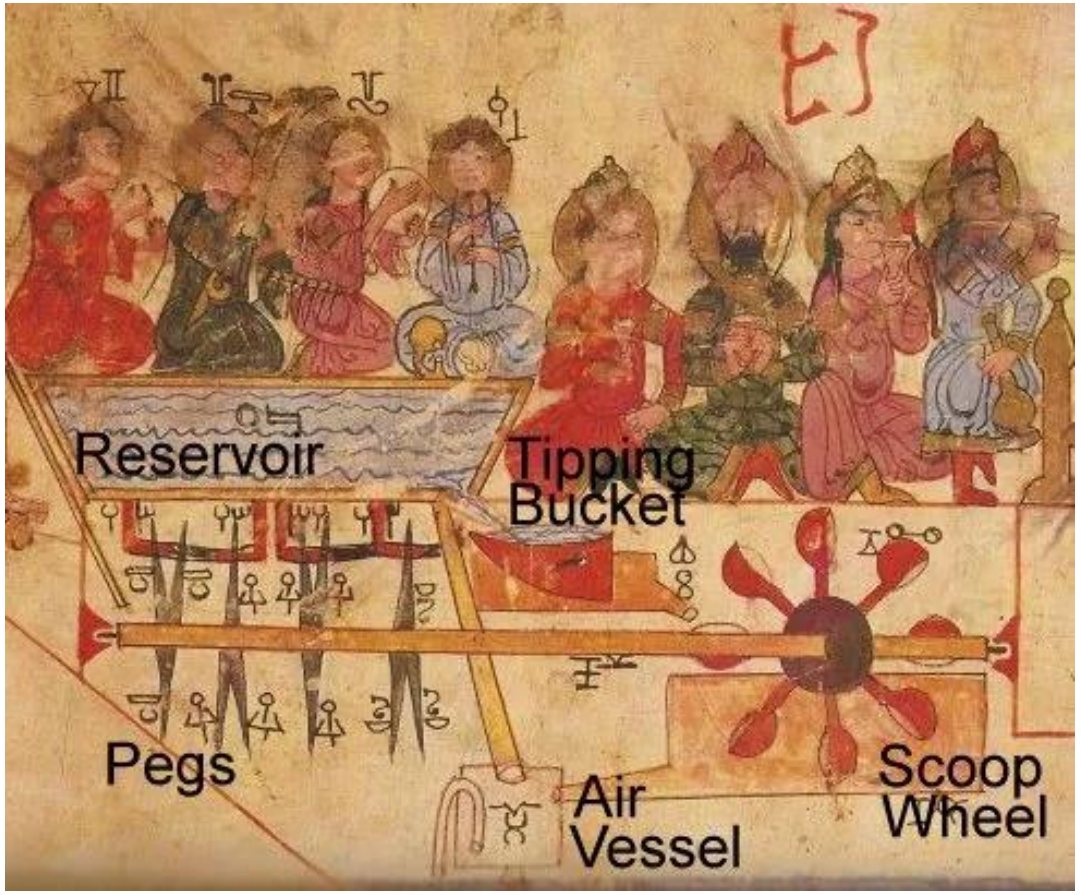
Al-Jazarī Creates the First Recorded Designs of a Programmable Automaton

1206

Automaton [Latin] : "acting of one's own will"

"**Musical Band**": programmable drum machine with pegs that bump into little levers that operate the percussion.

The drummer could be made to play different rhythms and drum patterns if the pegs were moved around.



Father of Mathematical Logic

David Hilbert

David Hilbert posed three fundamental problems about Mathematics in 1928: **(1) is mathematics complete; (2) is mathematics consistent; and (3) is mathematics decidable.**

Father of Mathematical Logic

David Hilbert

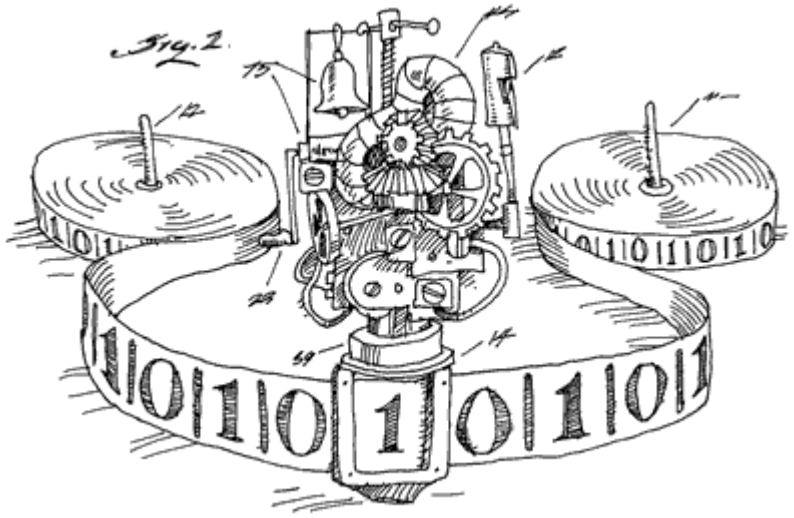
David Hilbert posed three fundamental problems about Mathematics in 1928: **(1) is mathematics complete; (2) is mathematics consistent; and (3) is mathematics decidable.**

Entscheidungsproblem (Decision Problem): Does there exist a definite method (or, *algorithm*) that can be applied to any mathematical assertion, and which is guaranteed to produce a correct decision as to whether that assertion is true.

Alan Turing: The Father of Modern Computer Science and Artificial Intelligence

On computable numbers, with an application to the Entscheidungsproblem

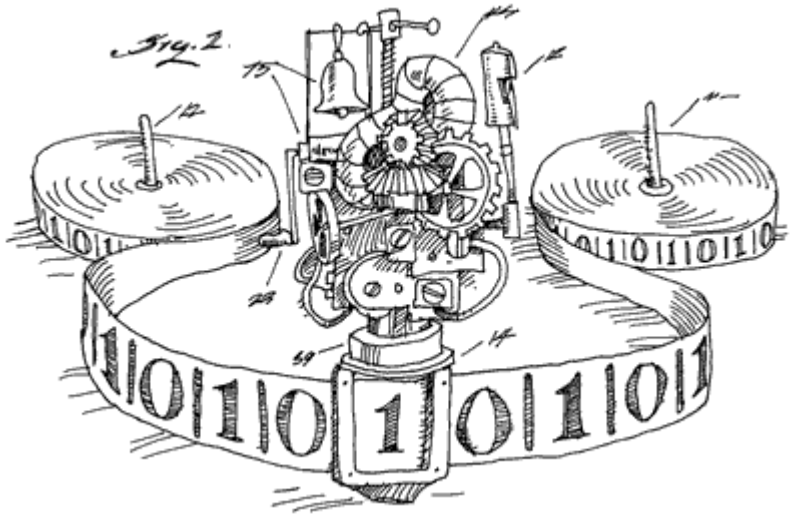
(Proc. Lond. Math. Soc., series 2 vol. **42** (1937), pp. 230–265)



Turing Machine – A Modern Automaton

On computable numbers, with an application to the Entscheidungsproblem

(Proc. Lond. Math. Soc., series 2 vol. **42** (1937), pp. 230–265)



A mathematical model of a hypothetical **computing machine** which can use a **predefined set of rules** to **determine a result** from a set of input variables.

Basically, an idealized model of a **central processing unit** (CPU).

Turing Machine – A Modern Automaton

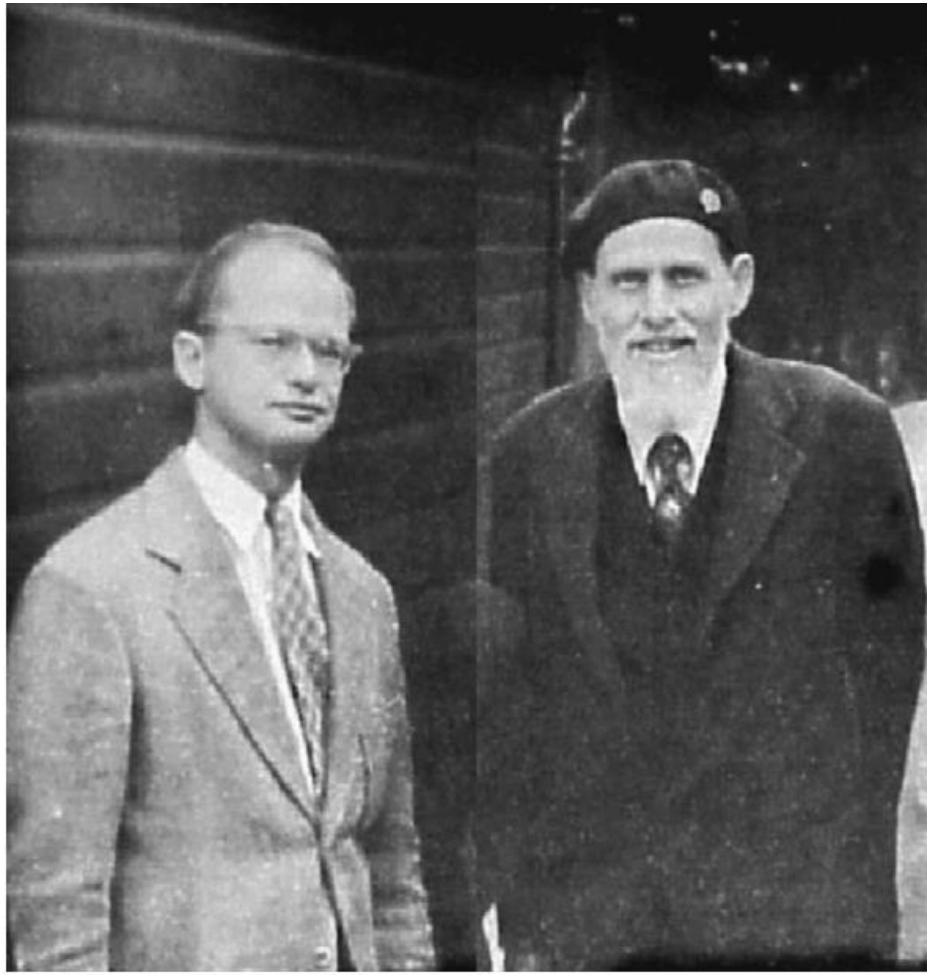
Turing Test

Turing on AI

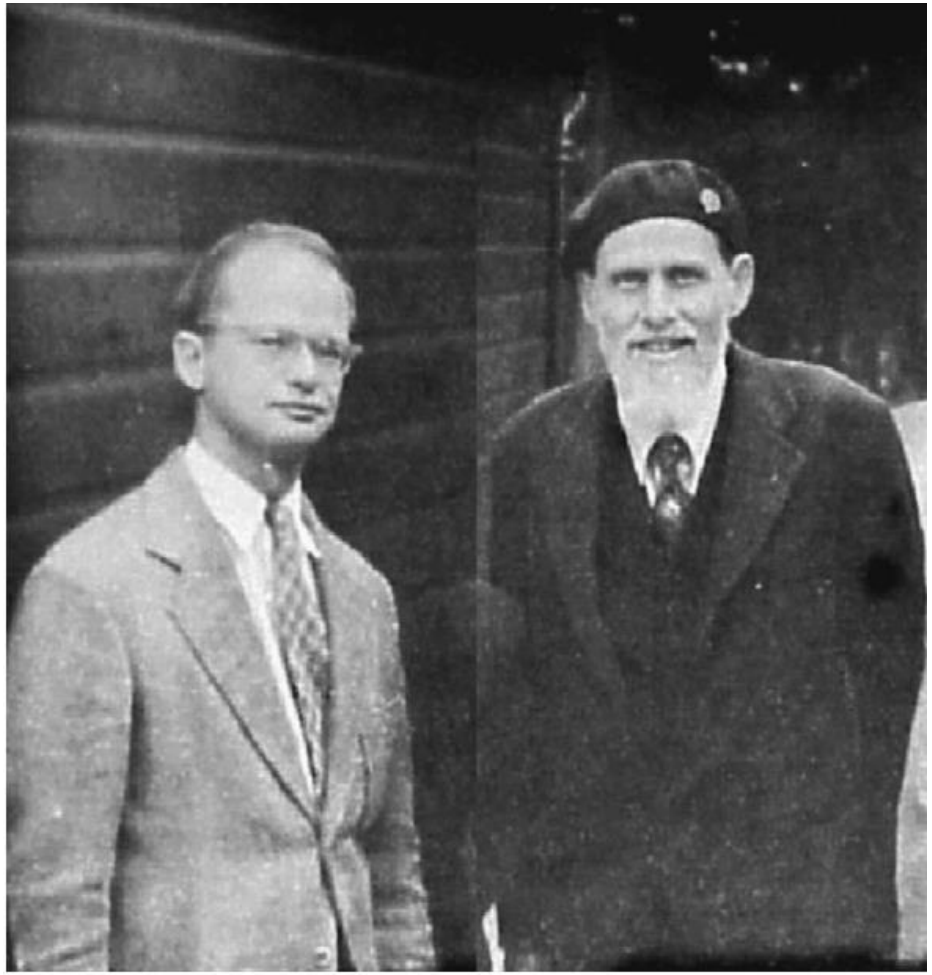
Turing Test

"A computer would deserve to be called intelligent if it could deceive a human into believing that it was human."

Turing on AI

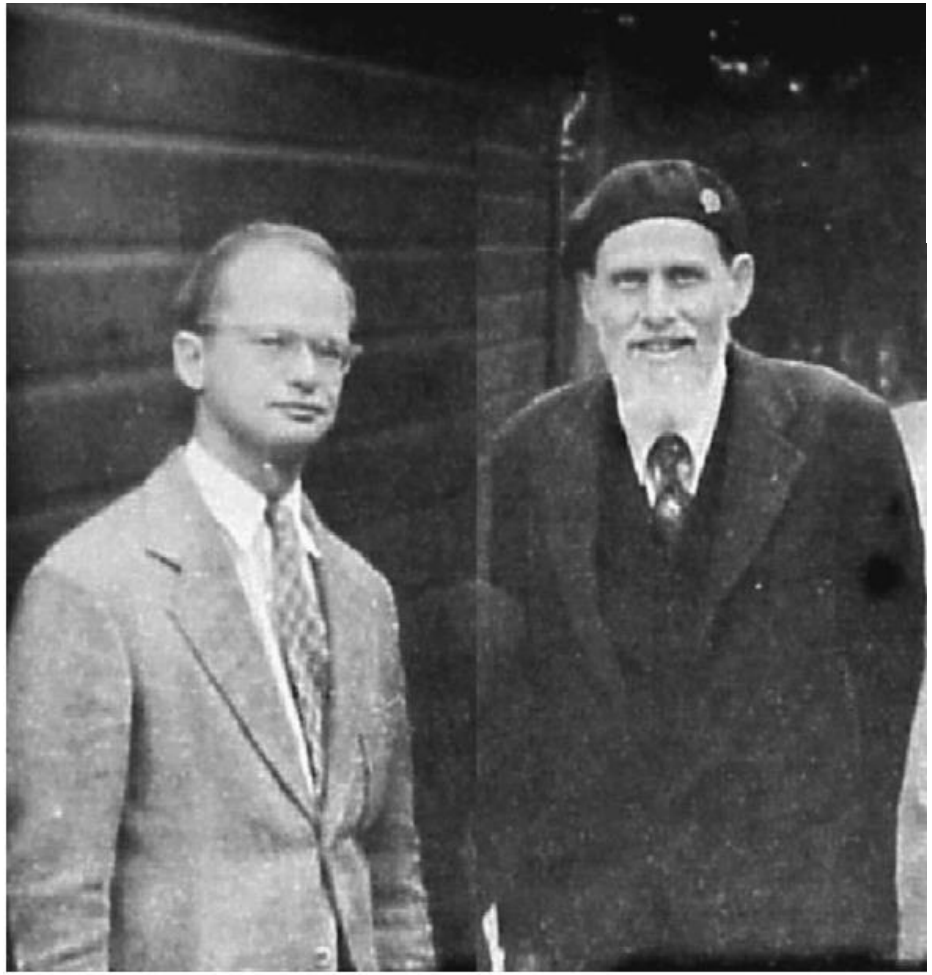


A Mathematician and a Neurophysiologist walk into a bar…



A Mathematician and a Neurophysiologist walk into a bar…

Having read Turing's work, they showed that a Turing Machine can be built using model of a human neuron.



A Mathematician and a Neurophysiologist walk into a bar…

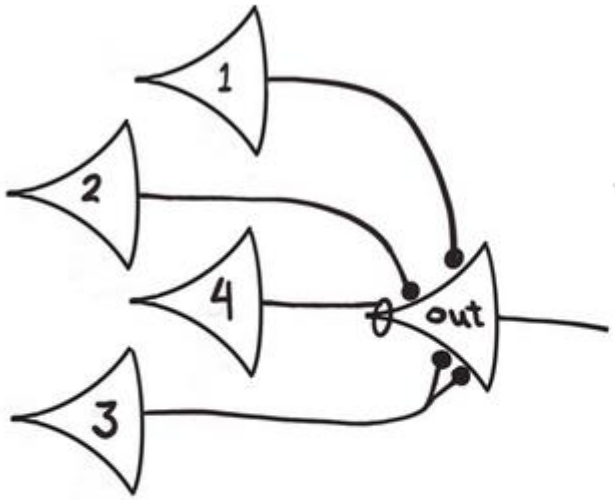
A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY*

- WARREN S. MCCULLOCH AND WALTER PITTS

Having read Turing's work, they showed that a Turing Machine can be built using model of a human neuron.

McCulloch & Pitts Publish the First Mathematical Model of a Neural Network

1943



t_i				t_{i+1}
1	2	3	4	out
F	F	F	F	F
T	F	F	F	F
F	T	F	F	F
T	T	F	F	T
F	F	T	F	T
T	F	T	F	T
F	T	T	F	T
T	T	T	F	T
X	X	X	T	F

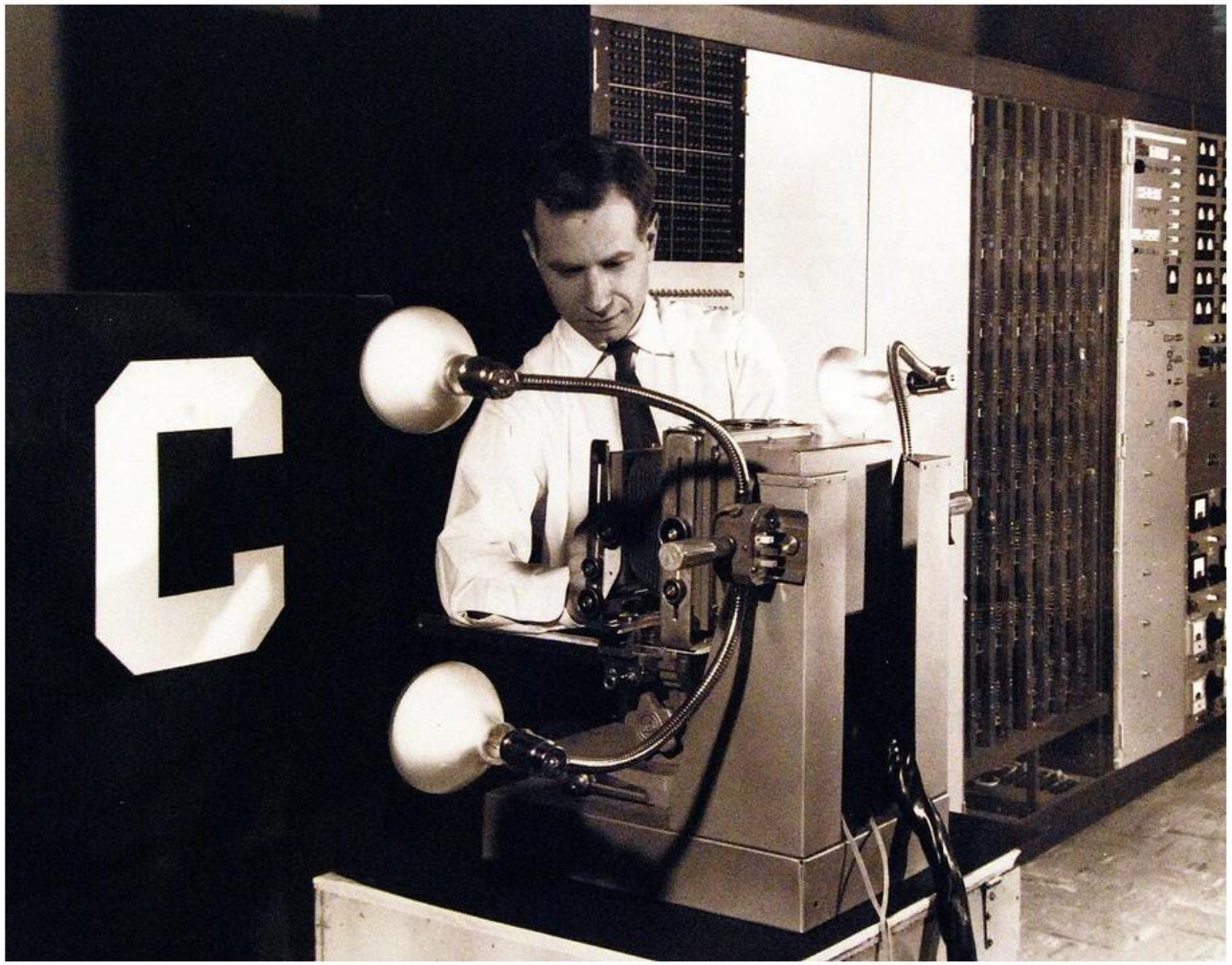
$$N_{out}(t+1) = ((N_1(t) \cdot N_2(t)) \vee N_3(t)) \cdot \sim N_4(t)$$

A simple neuronal network in McCullogh and Pitts notation, and its truth table. The state of each neuron is indicated by N_i.

MP Neuron does not include learning.

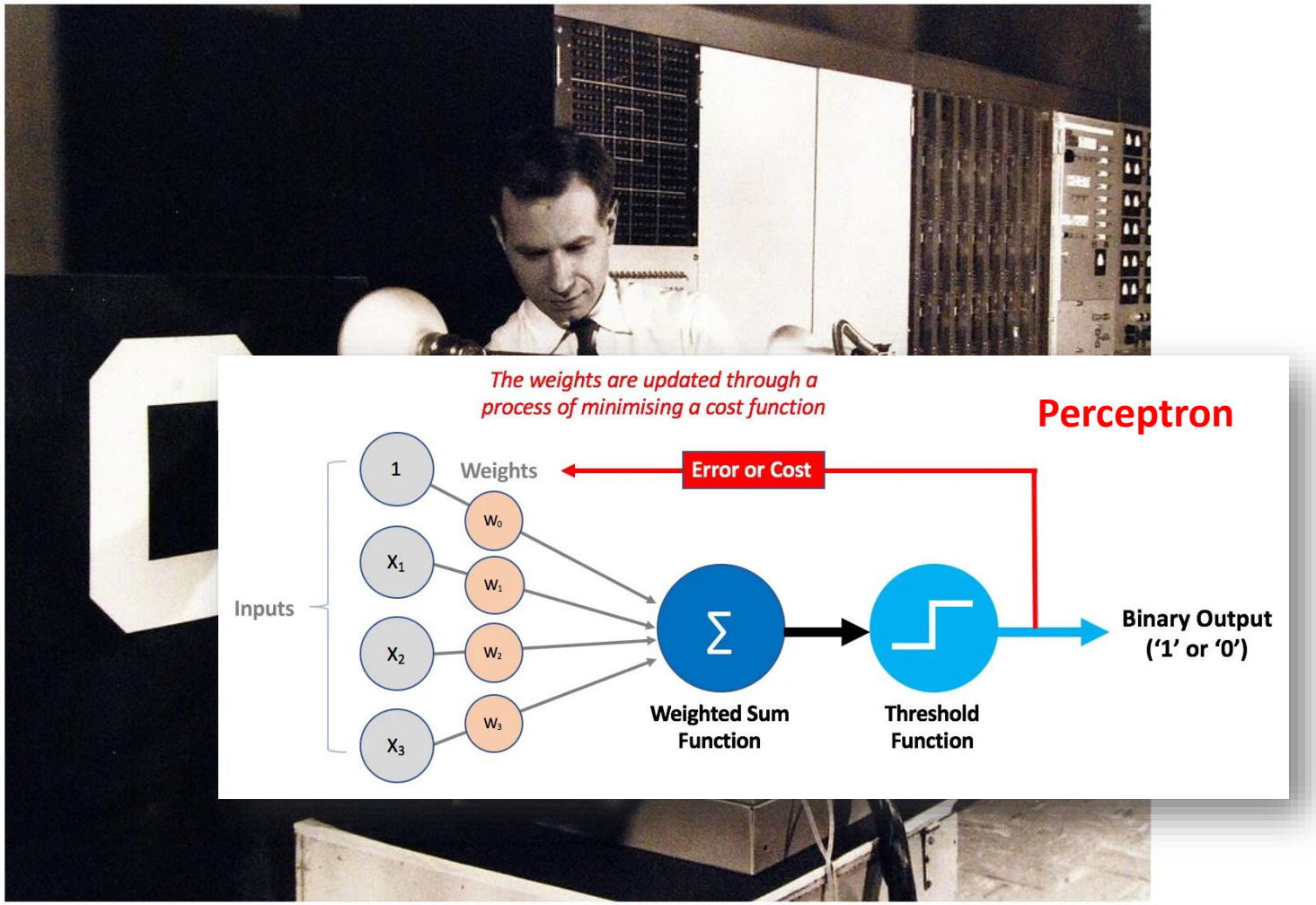
This “Logical Calculus” paper was widely read. Frank Rosenblatt used the ideas to build the ‘Mark 1 Perceptron’, a machine designed to perform image recognition.

This “Logical Calculus” paper was widely read. Frank Rosenblatt used the ideas to build the ‘Mark 1 Perceptron’, a machine designed to perform image recognition.



Camera system of the Mark 1 Perceptron with Rosenblatt’s Perceptron itself behind

This “Logical Calculus” paper was widely read. Frank Rosenblatt used the ideas to build the ‘Mark 1 Perceptron’, a machine designed to perform image recognition.

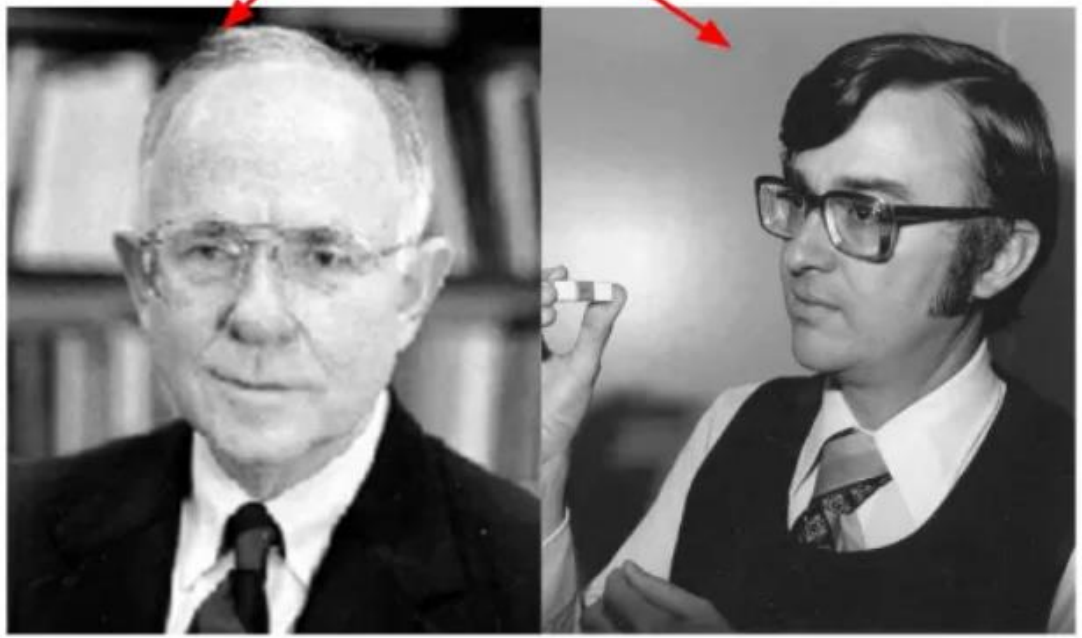


Camera system of the Mark 1 Perceptron with Rosenblatt’s Perceptron itself behind

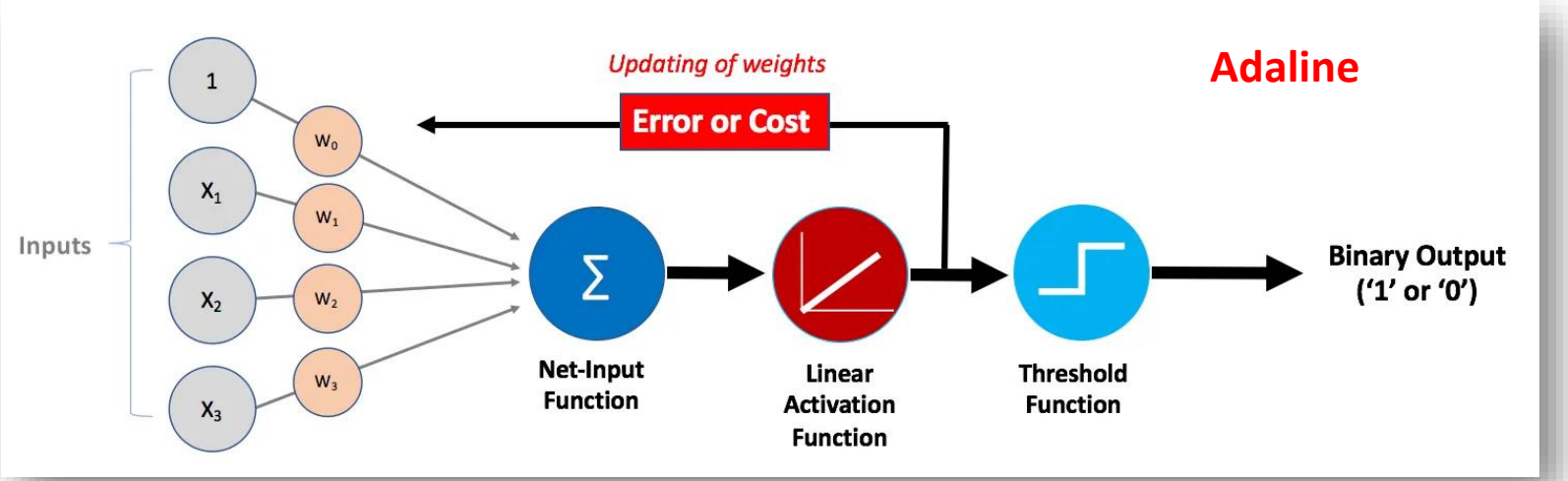
Included learning (as strengthening of neural pathways as “learning” in humans had recently been discovered in 1949)

Widrow and Hoff Decided to further improve on the Perceptron!

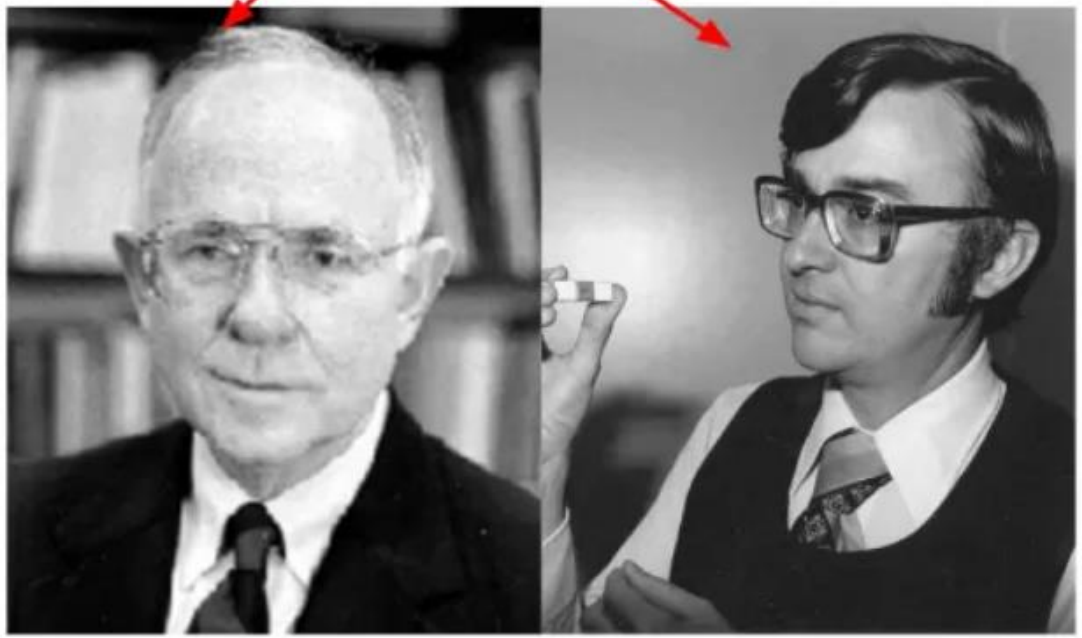
Widrow-Hoff - 1960



Widrow and Hoff Decided to further improve on the Perceptron!

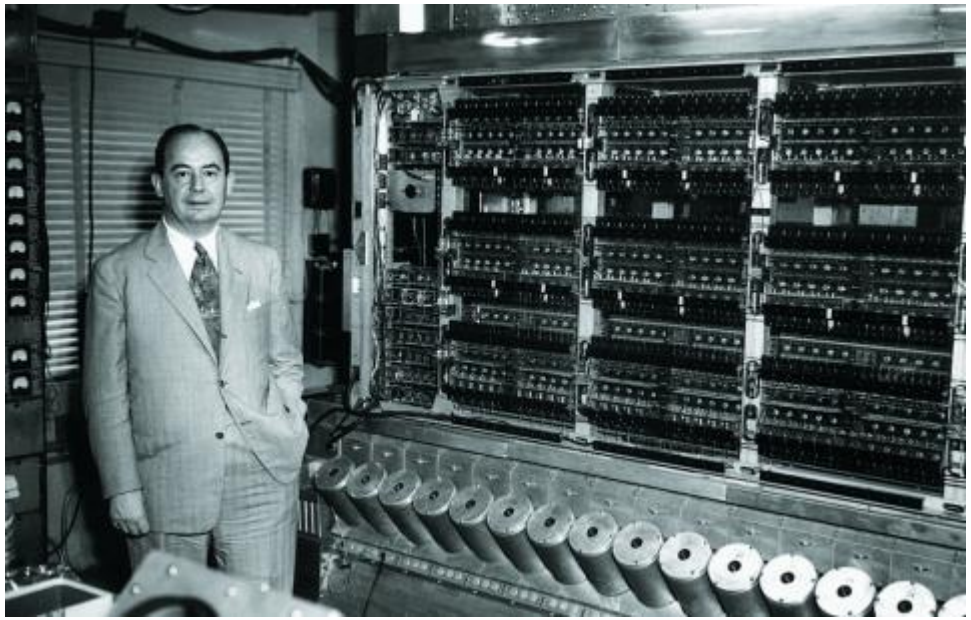


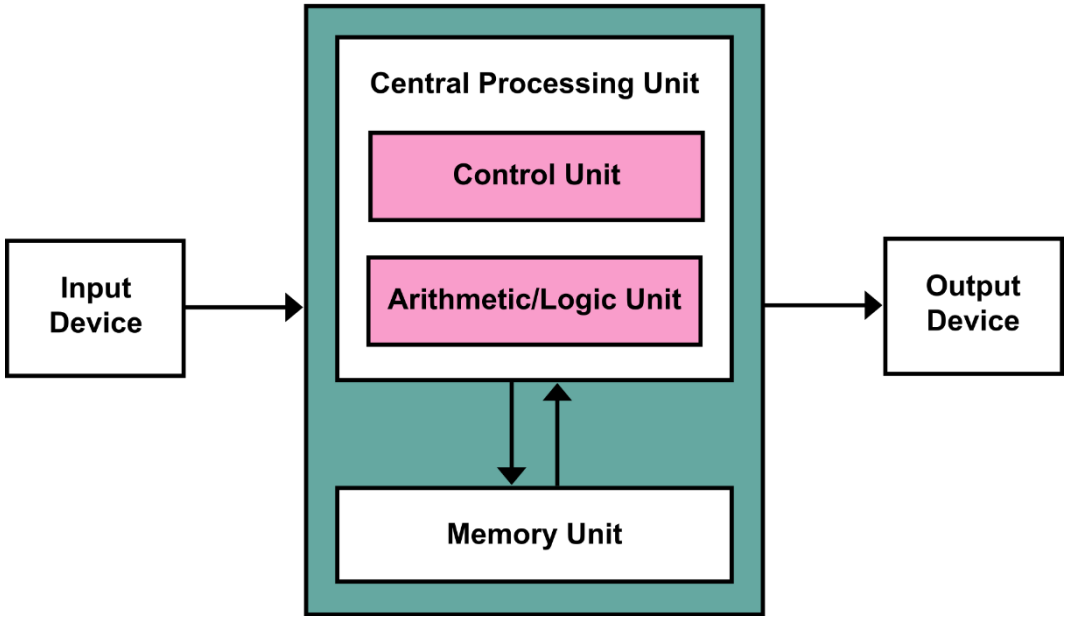
Widrow-Hoff - 1960

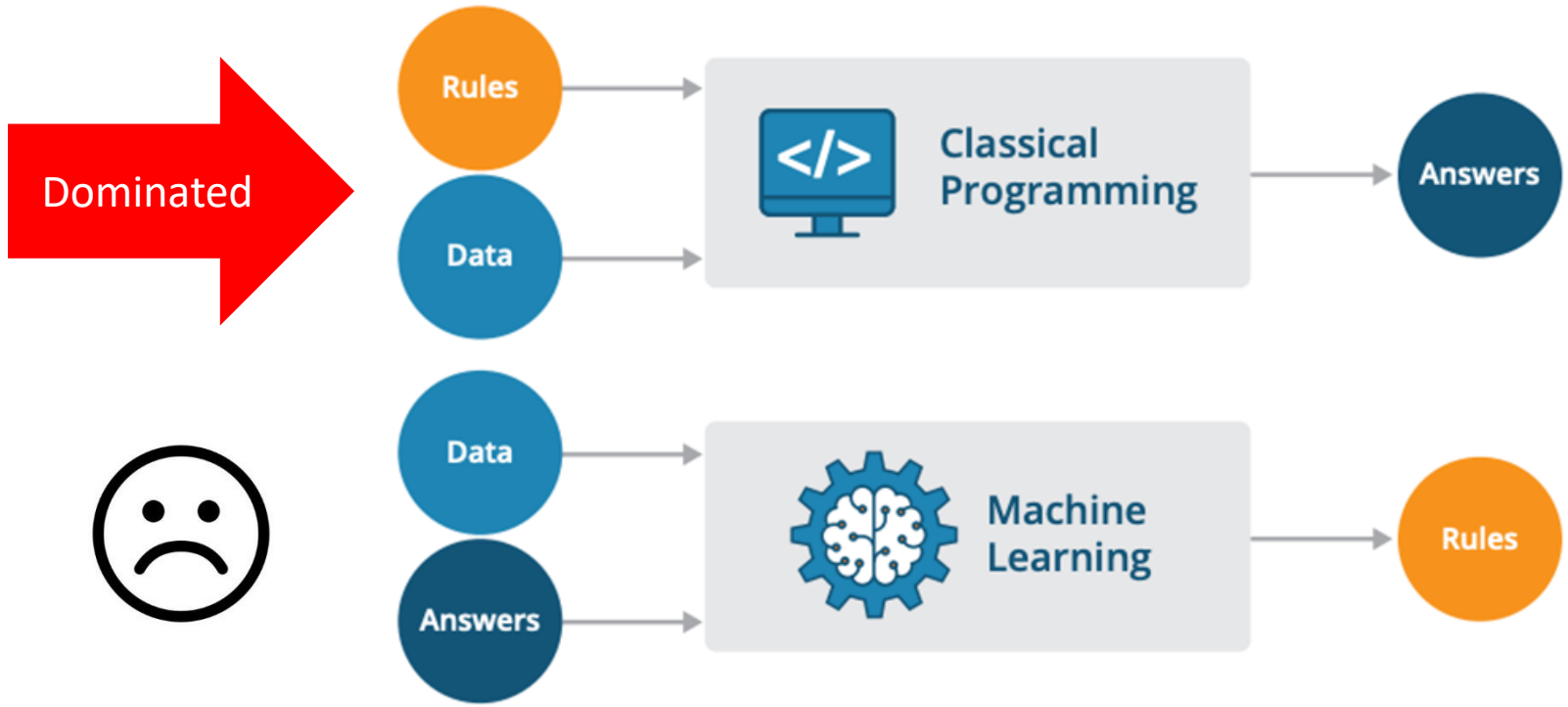


Unfortunately, technology of the time was more ready for the development of the “stored program” computer model by Von Neumann – who was also trying to implement Turing Machine.

Unfortunately, technology of the time was more ready for the development of the “stored program” computer model by Von Neumann – who was also trying to implement Turing Machine.






Dominated
Rules
Data
Classical Programming
Answers
Data
Answers
Machine Learning
Rules

1969
ReLU (rectified linear unit) activation function introduced (ReLU is currently the most popular activation function for deep learning).

1972-1999
RNNs developed and gradually improved.

1975
First multilayered neural network developed.

1979 – 1988
CNNs introduced and gradually improved.

1982
Joint US-Japan conference on Cooperative/Competitive Neural Networks.

1986
Multilayered Neural Networks gain some popularity, and finally backpropagation introduced to train them.

1969
ReLU (rectified linear unit) activation function introduced (ReLU is currently the most popular activation function for deep learning).

1972-1999
RNNs developed and gradually improved.

1975
First multilayered neural network developed.

1979 – 1988
CNNs introduced and gradually improved.

1982
Joint US-Japan conference on Cooperative/Competitive Neural Networks.

1986
Multilayered Neural Networks gain some popularity, and finally backpropagation introduced to train them.

Turn of Century – NNs ready, but where's the data and processing power?

Things slowed down again…

2009 – 2016

- Fast resurgence of ANNs with availability of unprecedented amounts of data (smart devices, social media, digital commerce) and high performance hardware.
- ANNs win many competitions and news coverage increases
- Concepts of Generative AI introduced.

2017

- Huge breakthrough as paper "Attention is all you need" introduces the next generation of ML called "Transformers"
- AI gains immense popularity in industry and academia.

2009 – 2016

- Fast resurgence of ANNs with availability of unprecedented amounts of data (smart devices, social media, digital commerce) and high performance hardware.
- ANNs win many competitions and news coverage increases
- Concepts of Generative AI introduced.

2017

- Huge breakthrough as paper "Attention is all you need" introduces the next generation of ML called "Transformers"
- AI gains immense popularity in industry and academia.

2023

ChatGPT takes the world by surprise and make AI a household word.

Next Stop…

python

Guido van Rossum

What makes Python ideal for AI and ML?

- High-level object-oriented language
- Interpreted and integrated
- Easy to learn, code, and read
- Resembles the English language
- Free and open-source
- Extensible
- Uses scripting to connect with existing components
- Dynamic typing and dynamic binding
- Cross-platform
- Great community support
- Simple syntax and indentation procedure
- Easy to debug
- Large number of libraries and frameworks

Turing

Parameter	Python	Java	JavaScript	C++
Runtime	Slower	Faster	Faster than Python	Slower
Length of code	Very short	5-10 times longer than Python	Short but longer than Python	5-10 times longer than Python
Community support for AI/ML	Large community	Small community	Small community	Small community
Syntax	Easy to write and similar to the English language	Difficult to master and involves brackets, unlike Python	Difficult to master compared to Python	Difficult to master and involves brackets, unlike Python

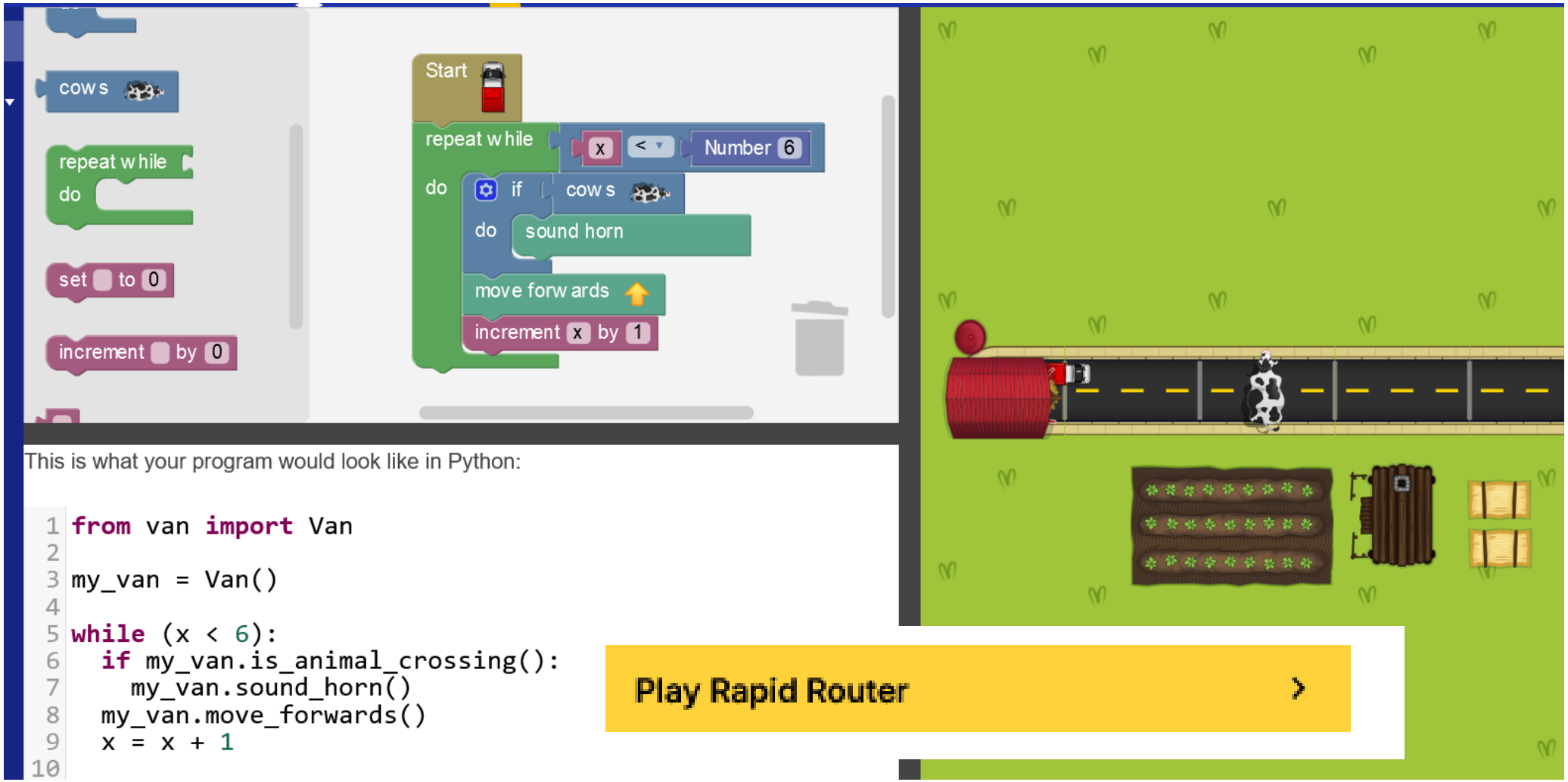
Coding is a Sense!

If no idea of coding yet, start here…

Block Coding

If no idea of coding yet, start here…

https://www.codeforlife.education



Block Coding

Next, try simple Python tasks…

https://www.codeforlife.education

Next, try simple Python tasks…

https://www.codeforlife.education

Python programming

The Python Den is an exploration of Python programming through a comprehensive course with free lesson plans, videos and worksheets to support you in your learning.

From foundational syntax to advanced concepts like loops and data manipulation, each session is a new challenge.

We aim to provide a tried-and-tested, structured set of lessons that you can use at home or in class, no matter your own level of experience.

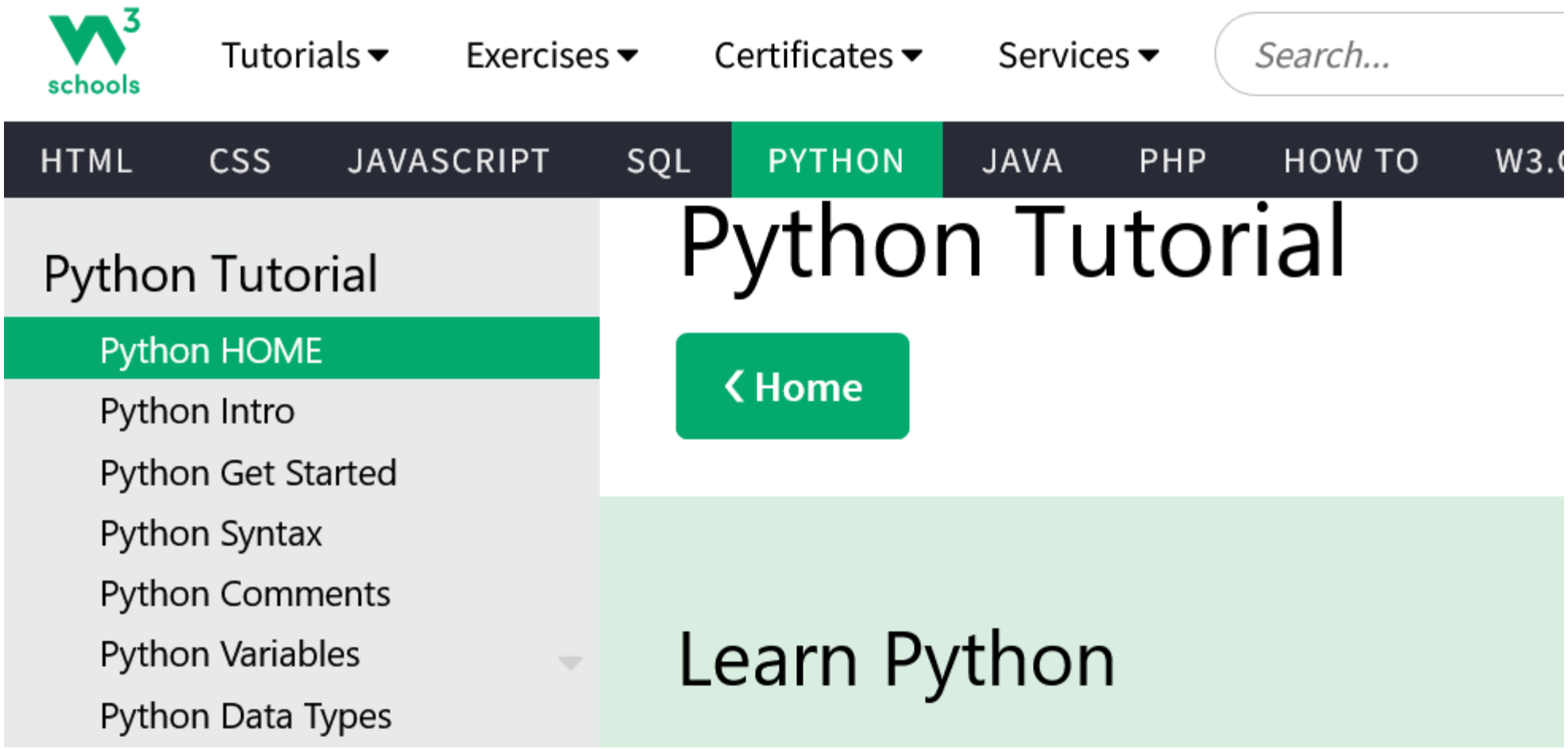
Play Python Den ›

Afterwards, go through Python tutorials…

https://www.w3schools.com/

Afterwards, go through Python tutorials…

https://www.w3schools.com/



Installing Python

Download

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

Latest: Python 3.12.6

Or, initially use an online version (though installing recommended)

Key Aspects of Learning Any Language

Key Aspects of Learning Any Language

- Input/Output
- Data Handling
- Data Manipulation
- Logic Flow Control
- Efficient Coding
- Help & Practice

Input/Output

Simple Input/Output

```python
print("Hello, World!")
```

```python
x = "Python"
y = "is"
z = "awesome"
print(x, y, z)
```

Create an f-string:

```python
age = 36
txt = f"My name is John, I am {age}"
print(txt)
```

Input/Output

Simple Input/Output

```
print("Hello, World!")
```

```
x = "Python"
y = "is"
z = "awesome"
print(x, y, z)
```

Create an f-string:

```
age = 36
txt = f"My name is John, I am {age}"
print(txt)
```

Get the characters from position 2 to position 5 (not included):

```
b = "Hello, World!"
print(b[2:5])
```

```
txt = f"The price is {20 * 59} dollars"
print(txt)
```

```
username = input("Enter username:")
print("Username is: " + username)
```

```
a = "Hello, World!"
print(a[1])
```

Input/Output

Working With Files

```
f = open("demofile.txt")
```

```
f = open("D:\\myfiles\welcome.txt", "r")
print(f.read())
```

Input/Output

Working With Files

```
f = open("demofile.txt")
```

```
f = open("D:\\myfiles\welcome.txt", "r")
print(f.read())
```

```
f = open("demofile.txt", "r")
print(f.read(5))
```

```
f = open("demofile.txt", "r")
print(f.readline())
print(f.readline())
```

```
f = open("demofile.txt", "r")
print(f.readline())
f.close()
```

Input/Output

Editing Files and Folders

Create a file called "myfile.txt":

```
f = open("myfile.txt", "x")
```

```
f = open("demofile2.txt", "a")
f.write("Now the file has more content!")
f.close()
```

```
f = open("demofile3.txt", "w")
f.write("Woops! I have deleted the content!")
f.close()
```

Input/Output

Editing Files and Folders

Create a file called "myfile.txt":

```
f = open("myfile.txt", "x")
```

```
f = open("demofile2.txt", "a")
f.write("Now the file has more content!")
f.close()
```

```
f = open("demofile3.txt", "w")
f.write("Woops! I have deleted the content!")
f.close()
```

```
import os
os.remove("demofile.txt")
```

Remove the folder "myfolder":

```
import os
os.rmdir("myfolder")
```

Data Handling

Single Variables & Collections of Variables

```python
x = 5
y = "John"
print(x)
print(y)
```

Data Handling

Single Variables & Collections of Variables

```python
x = 5
y = "John"
print(x)
print(y)
```

```python
fruits = ["apple", "banana", "cherry"]
x, y, z = fruits
print(x)
print(y)
print(z)
```

```python
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)
```

Data Handling

Single Variables & Collections of Variables

Text Type:	`str`
Numeric Types:	`int`, `float`, `complex`
Sequence Types:	`list`, `tuple`, `range`
Mapping Type:	`dict`
Set Types:	`set`, `frozenset`
Boolean Type:	`bool`
Binary Types:	`bytes`, `bytearray`, `memoryview`
None Type:	`NoneType`

Data Handling

Single Variables & Collections of Variables

Text Type:	`str`
Numeric Types:	`int`, `float`, `complex`
Sequence Types:	`list`, `tuple`, `range`
Mapping Type:	`dict`
Set Types:	`set`, `frozenset`
Boolean Type:	`bool`
Binary Types:	`bytes`, `bytearray`, `memoryview`
None Type:	`NoneType`

Deterministic vs. Random

```python
x = 5
print(type(x))
```

```python
import random

print(random.randrange(1, 10))
```

Data Handling

Single Variables & Collections of Variables

Example	Data Type
`x = "Hello World"`	str
`x = 20`	int
`x = 20.5`	float
`x = 1j`	complex
`x = ["apple", "banana", "cherry"]`	list
`x = ("apple", "banana", "cherry")`	tuple
`x = range(6)`	range
`x = {"name" : "John", "age" : 36}`	dict
`x = {"apple", "banana", "cherry"}`	set
`x = frozenset({"apple", "banana", "cherry"})`	frozenset
`x = True`	bool

Data Handling

Single Variables & Collections of Variables

```
x = str(3)    # x will be '3'
y = int(3)    # y will be 3
z = float(3)  # z will be 3.0
```

Example	Data Type
x = "Hello World"	str
x = 20	int
x = 20.5	float
x = 1j	complex
x = ["apple", "banana", "cherry"]	list
x = ("apple", "banana", "cherry")	tuple
x = range(6)	range
x = {"name" : "John", "age" : 36}	dict
x = {"apple", "banana", "cherry"}	set
x = frozenset({"apple", "banana", "cherry"})	frozenset
x = True	bool

Data Handling

Single Variables &
Collections of Variables

Data Handling

Single Variables &
Collections of Variables

List

General purpose
Most widely used data structure
Grow and shrink size as needed
Sequence type
Sortable

Tuple

Immutable (can't add/change)
Useful for fixed data
Faster than Lists
Sequence type

Set

Store non-duplicate items
Very fast access vs Lists
Math Set ops (union, intersect)
Unordered

Dict

Key/Value pairs
Associative array, like Java HashMap
Unordered

- **List** is a collection which is ordered and changeable. Allows duplicate members.
- **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
- **Set** is a collection which is unordered, unchangeable*, and unindexed. No duplicate members.
- **Dictionary** is a collection which is ordered** and changeable. No duplicate members.

Data Handling

A list with strings, integers and boolean values:

```
list1 = ["abc", 34, True, 40, "male"]
```

- **List** is a collection which is ordered and changeable. Allows duplicate members.
- **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
- **Set** is a collection which is unordered, unchangeable*, and unindexed. No duplicate members.
- **Dictionary** is a collection which is ordered** and changeable. No duplicate members.

Data Handling

A list with strings, integers and boolean values:

```
list1 = ["abc", 34, True, 40, "male"]
```

A tuple with strings, integers and boolean values:

```
tuple1 = ("abc", 34, True, 40, "male")
```

- **List** is a collection which is ordered and changeable. Allows duplicate members.
- **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
- **Set** is a collection which is unordered, unchangeable*, and unindexed. No duplicate members.
- **Dictionary** is a collection which is ordered** and changeable. No duplicate members.

Data Handling

A list with strings, integers and boolean values:

```
list1 = ["abc", 34, True, 40, "male"]
```

A tuple with strings, integers and boolean values:

```
tuple1 = ("abc", 34, True, 40, "male")
```

Duplicate values will be ignored:

```
thisset = {"apple", "banana", "cherry", "apple"}

print(thisset)
```

- **List** is a collection which is ordered and changeable. Allows duplicate members.
- **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
- **Set** is a collection which is unordered, unchangeable*, and unindexed. No duplicate members.
- **Dictionary** is a collection which is ordered** and changeable. No duplicate members.

Data Handling

A list with strings, integers and boolean values:

```
list1 = ["abc", 34, True, 40, "male"]
```

A tuple with strings, integers and boolean values:

```
tuple1 = ("abc", 34, True, 40, "male")
```

Duplicate values will be ignored:

```
thisset = {"apple", "banana", "cherry", "apple"}

print(thisset)
```

Print the "brand" value of the dictionary:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict["brand"])
```

Data Handling

Single Variables & **Collections** of Variables

Print the second item of the list:

```
thislist = ["apple", "banana", "cherry"]
print(thislist[1])
```

```
thislist = ["apple", "banana", "cherry", "apple", "cherry"]
print(thislist)
```

Change the second item:

```
thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"
print(thislist)
```

Data Handling

Single Variables & **Collections** of Variables

Print the second item of the list:

```
thislist = ["apple", "banana", "cherry"]
print(thislist[1])
```

```
thislist = ["apple", "banana", "cherry", "apple", "cherry"]
print(thislist)
```

Change the second item:

```
thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"
print(thislist)
```

Convert the tuple into a list to be able to change it:

```
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)

print(x)
```

Operators & Methods

Data Manipulation

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators

Operators & Methods

Data Manipulation

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators

Arithmetic operators are used with numeric values to perform common mathematical operations:

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Operators & Methods

Data Manipulation

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators

Python Assignment Operators

Assignment operators are used to assign values to variables:

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3

Operators & Methods

Data Manipulation

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators

Python Comparison Operators

Comparison operators are used to compare two values:

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Data Manipulation

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators

Operators & Methods

Python Logical Operators

Logical operators are used to combine conditional statements:

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

Data Manipulation

Operators & Methods

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators

Python Membership Operators

Membership operators are used to test if a sequence is presented in an object:

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

Data Manipulation

Operators & **Methods**

Python Lists

- Access List Items
- Change List Items
- Add List Items
- Remove List Items
- Loop Lists
- List Comprehension
- Sort Lists
- Copy Lists
- Join Lists

Python Tuples

- Access Tuples
- Update Tuples
- Unpack Tuples
- Loop Tuples
- Join Tuples

Python Sets

- Access Set Items
- Add Set Items
- Remove Set Items
- Loop Sets
- Join Sets

Python Dictionaries

- Access Items
- Change Items
- Add Items
- Remove Items
- Loop Dictionaries
- Copy Dictionaries
- Nested Dictionaries

Data Manipulation

Operators & **Methods**

Method	Description
capitalize()	Converts the first character to upper case
casefold()	Converts string into lower case
center()	Returns a centered string
count()	Returns the number of times a specified value occurs in a string
encode()	Returns an encoded version of the string
endswith()	Returns true if the string ends with the specified value
expandtabs()	Sets the tab size of the string
find()	Searches the string for a specified value and returns the position of where it was found
format()	Formats specified values in a string
format_map()	Formats specified values in a string
index()	Searches the string for a specified value and returns the position of where it was found
isalnum()	Returns True if all characters in the string are alphanumeric
isalpha()	Returns True if all characters in the string are in the alphabet

Data Manipulation

Operators & Methods

Check if "free" is present in the following text:

```
txt = "The best things in life are free!"
print("free" in txt)
```

Check if "expensive" is NOT present in the following text:

```
txt = "The best things in life are free!"
print("expensive" not in txt)
```

The `len()` function returns the length of a string:

```
a = "Hello, World!"
print(len(a))
```

Data Manipulation

Operators & Methods

Check if "free" is present in the following text:

```
txt = "The best things in life are free!"
print("free" in txt)
```

Check if "expensive" is NOT present in the following text:

```
txt = "The best things in life are free!"
print("expensive" not in txt)
```

```
a = "Hello, World!"
print(a.replace("H", "J"))
```

To add a space between them, add a `" "`:

```
a = "Hello"
b = "World"
c = a + " " + b
print(c)
```

The `len()` function returns the length of a string:

```
a = "Hello, World!"
print(len(a))
```

The `split()` method splits the string into substrings if it finds instances of the separator:

```
a = "Hello, World!"
print(a.split(",")) # returns ['Hello', ' World!']
```

Logic Flow Control

Conditions & Loops

Logic Flow Control

Conditions & Loops

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

Logic Flow Control

Conditions & Loops

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

```
a = 200
b = 33
c = 500
if a > b and c > a:
  print("Both conditions are True")
```

Check if "apple" is present in the list:

```
thislist = ["apple", "banana", "cherry"]
if "apple" in thislist:
  print("Yes, 'apple' is in the fruits list")
```

Logic Flow Control

Conditions & Loops

```
i = 1
while i < 6:
  print(i)
  i += 1
```

Increment the sequence with 3 (default is 1):

```
for x in range(2, 30, 3):
  print(x)
```

Logic Flow Control

Conditions & Loops

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    break
  print(x)
```

```
i = 1
while i < 6:
  print(i)
  i += 1
```

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

Increment the sequence with 3 (default is 1):

```
for x in range(2, 30, 3):
  print(x)
```

```
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)

# Note that number 3 will not be printed as we skipped it
```

Efficient Coding

Functions, Classes, Modules, Libraries

Efficient Coding

Functions, Classes, Modules, Libraries

```
def my_function(x):
  return 5 * x

print(my_function(3))
print(my_function(5))
print(my_function(9))
```

Efficient Coding

Functions, Classes, Modules, Libraries

```
def get_user_data():
    return 'Anna', 23, 'anna123'


name, age, id = get_user_data()
print('Got the user data:', name, age, id)
# Got the user data: Anna 23 anna123
```

```
def my_function(x):
  return 5 * x

print(my_function(3))
print(my_function(5))
print(my_function(9))
```

```
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Emil", "Refsnes")
```

Multiply argument `a` with argument `b` and return the result:

```
x = lambda a, b : a * b
print(x(5, 6))
```

Efficient Coding

Functions, **Classes**, Modules, Libraries

Create a class named Person, use the __init__() function to assign values for name and age:

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

p1 = Person("John", 36)

print(p1.name)
print(p1.age)
```

Efficient Coding

Functions, Classes,
Modules, Libraries

Save this code in a file named `mymodule.py`

```
def greeting(name):
  print("Hello, " + name)
```

```
import mymodule

mymodule.greeting("Jonathan")
```

User-Defined
Modules

Efficient Coding

Functions, Classes, **Modules, Libraries**

Save this code in a file named `mymodule.py`

```
def greeting(name):
  print("Hello, " + name)
```

```
import mymodule

mymodule.greeting("Jonathan")
```

User-Defined Modules

```
import math

x = math.sqrt(64)

print(x)
```

Predefined Modules

Search the string to see if it starts with "The" and ends with "Spain":

```
import re

txt = "The rain in Spain"
x = re.search("^The.*Spain$", txt)
```

Efficient Coding

Functions, Classes,
Modules, Libraries

Python Modules

- NumPy Tutorial
- Pandas Tutorial
- SciPy Tutorial
- Django Tutorial

Python Matplotlib

Help & Practice

Tutorials, Help, Documentation, Forums

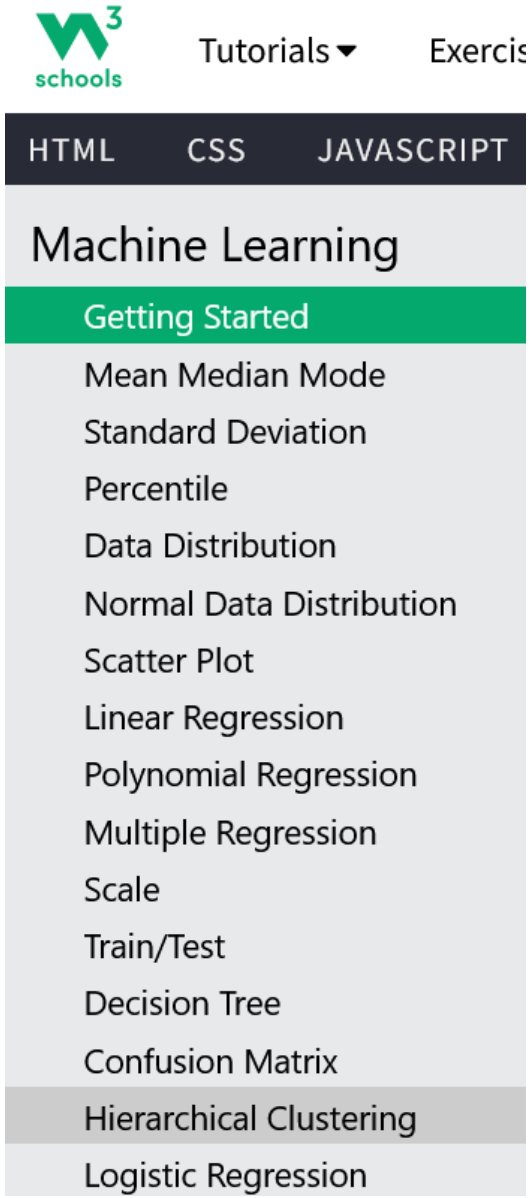
https://www.python.org/

https://www.w3schools.com/

```
help(print)
```

HW:

- Find Python Forums
- Take Python Basics Tutorials Online
- Take Python Machine Learning Tutorials Online
- Implement All the Mathematics Topics We Cover in Python



Questions?? Thoughts??