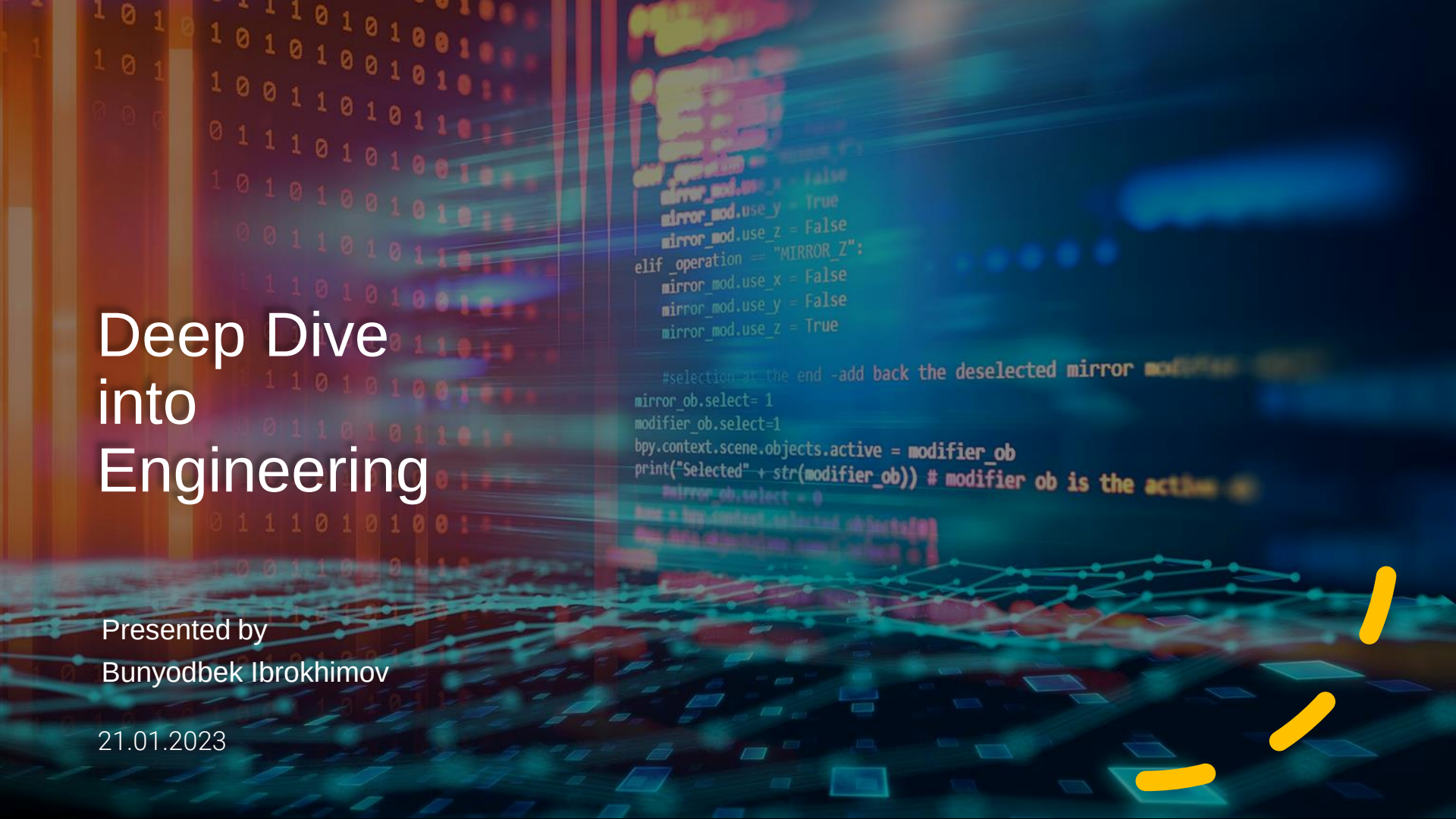


Deep Dive into Engineering

Presented by

Bunyodbek Ibrokhimov

21.01.2023

Agenda

01

What is software engineering?

02

Where to start:
A Roadmap

03

Different levels of software engineer

04

Tips to become a pro

Software Engineering

ChatGPT

Software engineering:
The design and development of software systems, including the use of programming languages, software development methodologies, and software testing.

From coding to ML engineer

Computer engineering:
Software Engineering +

- **Computer architecture**: The design and organization of computer systems, including the design of processors, memory systems, and other components.
- **Computer networks**: The design, implementation, and maintenance of computer networks and communication systems, including local area networks (LANs), wide area networks (WANs), and the internet.
- **Embedded systems**: The design and development of computer systems that are integrated into other devices or products, such as cars, appliances, and medical equipment.

Where to start: A Roadmap

Programming language

Python
- Syntax
- Basics (variables, data types, casting)
- Loops, iterators, scope, modules, function

Algorithms and data structures

- **Sequence of instructions**
- **A process or set of rules**

- Dictionaries
- Linked lists
- Binary search tree
- Arrays, Lists, Sets

Core ML/DL concepts

- Matrix multiplication
- Neural network structure (layers)
- Inference
- Backpropagation
- Activation functions
- Error functions
- Regularizations

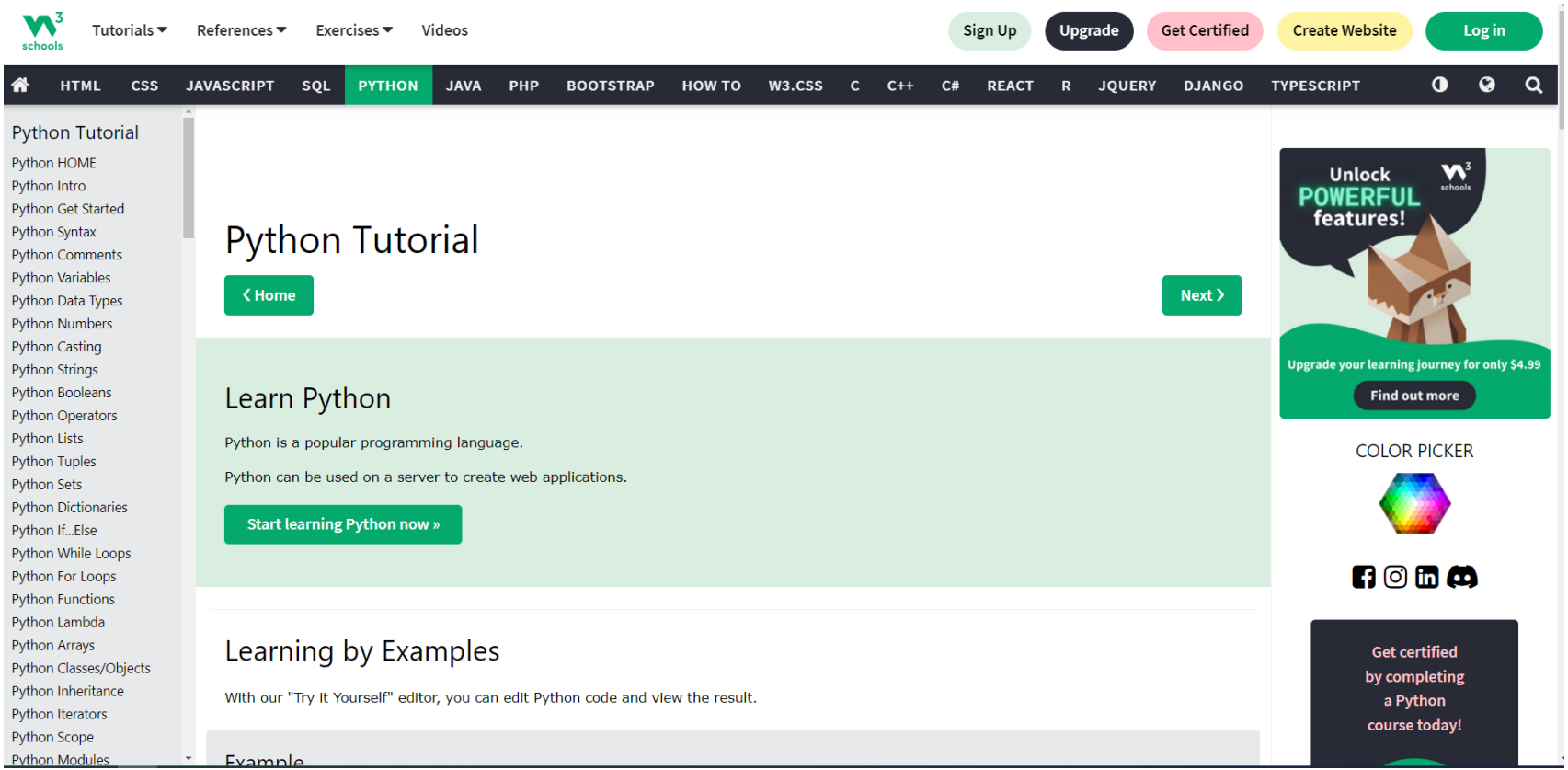
Tools and frameworks

- Pytorch
- Tensorflow
- Keras

- Pandas
- NumPy
- Matplotlib
- OpenCV
- ROS

- Git/GitLab
- Linux

Python


Tutorials
References
Exercises
Videos
Sign Up
Upgrade
Get Certified
Create Website
Log in
HTML CSS JAVASCRIPT SQL PYTHON JAVA PHP BOOTSTRAP HOW TO W3.CSS C C++ C# REACT R JQUERY DJANGO TYPESCRIPT
Python Tutorial
Python HOME
Python Intro
Python Get Started
Python Syntax
Python Comments
Python Variables
Python Data Types
Python Numbers
Python Casting
Python Strings
Python Booleans
Python Operators
Python Lists
Python Tuples
Python Sets
Python Dictionaries
Python If...Else
Python While Loops
Python For Loops
Python Functions
Python Lambda
Python Arrays
Python Classes/Objects
Python Inheritance
Python Iterators
Python Scope
Python Modules
Python Tutorial
❮ Home
Next ❯
Learn Python
Python is a popular programming language.
Python can be used on a server to create web applications.
Start learning Python now »
Learning by Examples
With our "Try it Yourself" editor, you can edit Python code and view the result.
Example
Unlock POWERFUL features!
Upgrade your learning journey for only $4.99
Find out more
COLOR PICKER
Get certified by completing a Python course today!

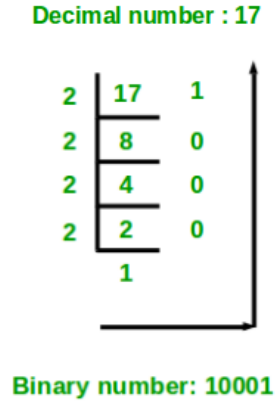
Python

- YouTube
- Coursera
- Udemy
- Codecademy
- Codefinity (from zero to hero)
- W3Schools

Algorithm

- **Covert Decimal (17) to binary (10001)**
- **Valid palindrome**
- **Remove zeros from string**
- **…**

1 read (number)
2 loop (number > 0)
  1 digit = number modulo 2
  2 print (digit)
  3 number = number / 2



test.cpp

```cpp
void decToBinary(int n){
    int binaryNum;
    int i = 0;
    while (n > 0) {
        binaryNum[i] = n % 2;
        n = n / 2;
        i++;
    }
    for (int j = i - 1; j >= 0; j--)
        cout << binaryNum[j];
}
int main(){
    int n = 17;
    decToBinary(n);
    return 0;
}
```

```python
def decToBinary(n):
    binaryNum = [0] * n
    i = 0
    while (n > 0):
        binaryNum[i] = n % 2
        n = int(n / 2)
        i += 1
    for j in range(i - 1, -1, -1):
        print(binaryNum[j], end = "")
n = 17
decToBinary(n)
```

Algorithm

- **Covert Decimal (17) to binary (10001)**
- **Valid palindrome**
- **Remove zeros from string**
- **…**

```
Input : malayalam
Output : Yes

Input : geeks
Output : No
```

Integer: 1221

1221 // 1000 == 1221 % 10
1221 // 100 == 1221 % 100
1221 // 10 == 1221 % 1000

127809721

```python
def isPalindrome(str):
    for i in range(0, int(len(str)/2)):
        if str[i] != str[len(str)-i-1]:
            return False
    return True

s = "malayalam"
ans = isPalindrome(s)

if (ans):print("Yes")
else:print("No")
```

```python
def isPalindrome(s):
    return s == s[::-1]

s = "malayalam"
ans = isPalindrome(s)

if ans:print("Yes")
else:print("No")
```

Time complexity: O(n)
Auxiliary Space: O(1)

Data structure

- **Search from sorted array**

Input: arr = [-1,0,3,5,9,12], target = 9
Output: 4

```python
arr = [-1,0,3,5,9,12]
target = 13
def search(target):
    for i in range(len(arr)):
        if arr[i] == target:
            return(i)
    return -1
print(search(target))
```

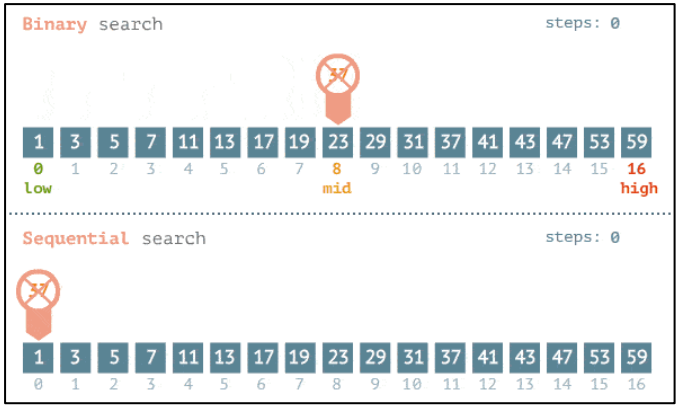
Time complexity: O(n)

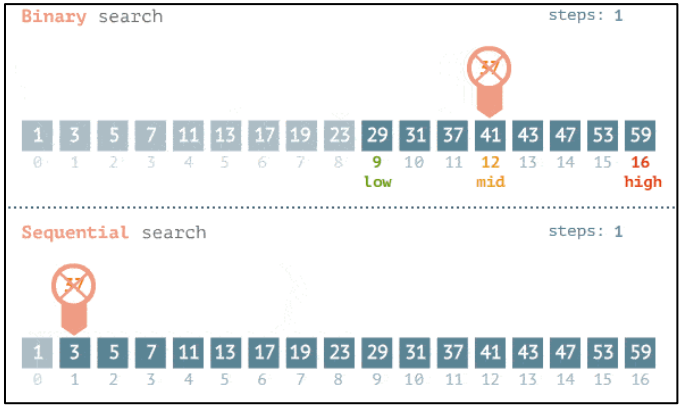
```python
arr = [-1,0,3,5,9,12]
target = 9
def search(self, arr, target):
    left = 0
    right = len(arr)-1
    while left<=right:
        mid = (left+right)//2
        if arr[mid]==target:
            return mid
        elif arr[mid]>target:
            right = mid-1
        else:
            left = mid+1
    return -1
```

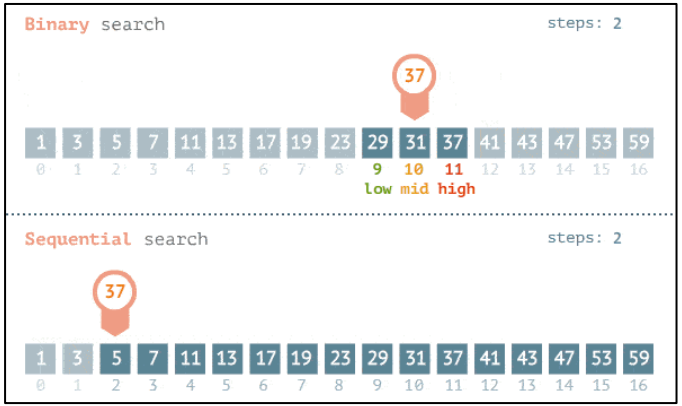
```python
arr = [-1,0,3,5,9,12]
target = 9
def search(arr, target):
    if not target in arr:
        return -1
    else:
        return arr.index(target)
```

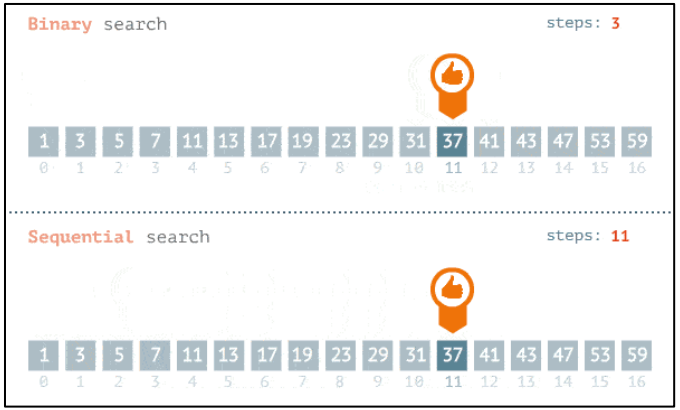
Time complexity: O(logn)

Data structure



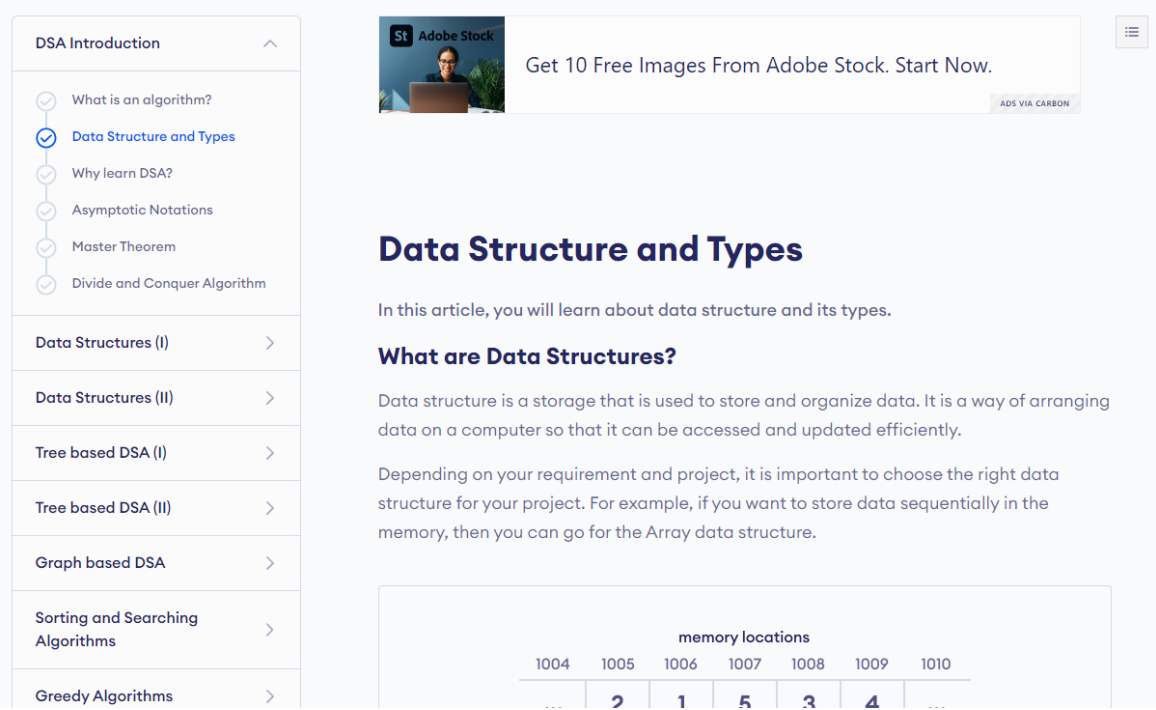






Data structure

https://www.programiz.com/dsa/data-structure-types



Where to start: A Roadmap

Programming language

Python
- Syntax
- Basics (variables, data types, casting)
- Loops, iterators, scope, modules, function

Algorithms and data structures

- **Sequence of instructions**
- **A process or set of rules**

- Dictionaries
- Linked lists
- Binary search tree
- Arrays, Lists, Sets
- Heap, stack
- Monotonic stack

Core ML/DL concepts

- Matrix multiplication
- Neural network structure (layers)
- Inference
- Backpropagation
- Activation functions
- Error functions
- Regularizations

Tools and frameworks

- Pytorch
- Tensorflow
- Keras

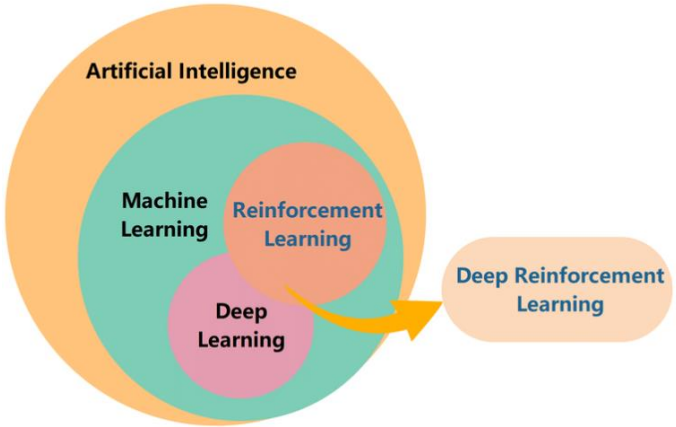
- Pandas
- NumPy
- Matplotlib
- OpenCV
- ROS

- Git/GitLab
- Linux

Core ML/DL concepts

- Matrix multiplication
- Neural network structure (layers)
- Inference
- Backpropagation
- Activation functions
- Error functions
- Regularizations

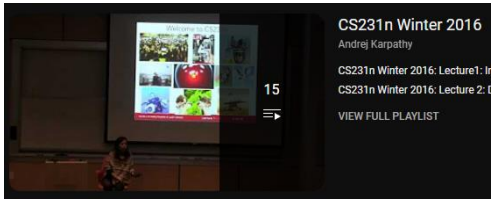
- ML solution types
- DNN types

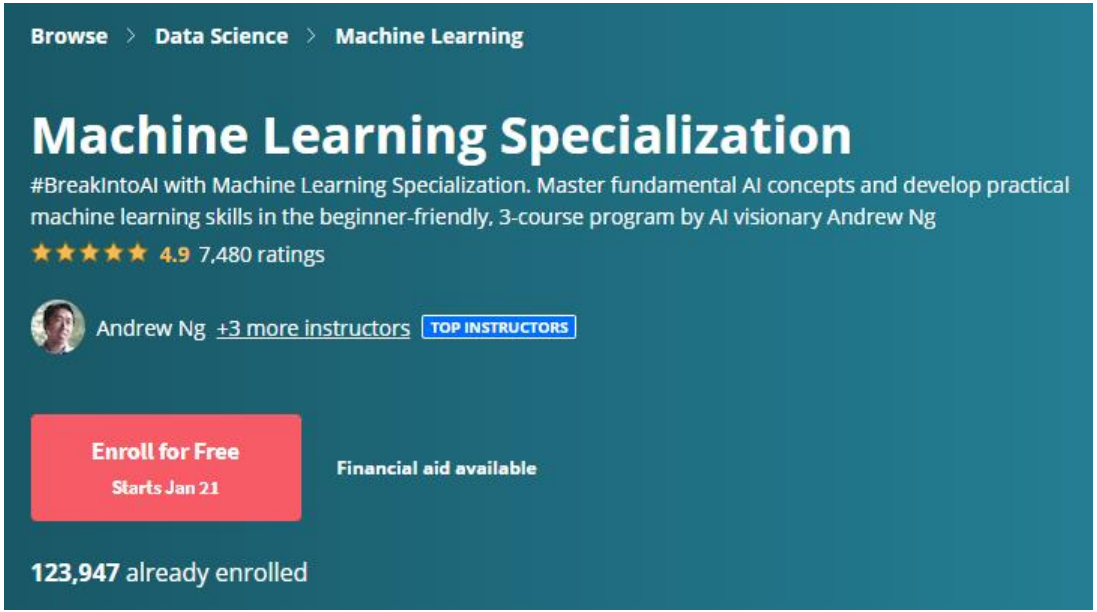


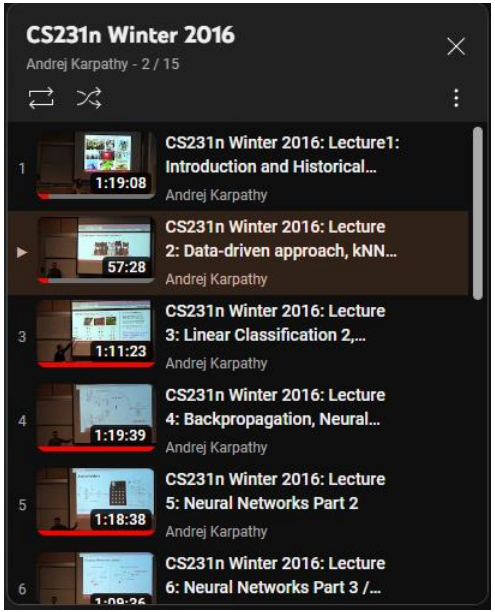
Core ML/DL concepts

Must-learn courses:

- Coursera: Machine learning by Andrew Ng (ML and DL concepts)
- YouTube: cs231n 2016 winter by Andrej Karpathy (theory, impl., types)

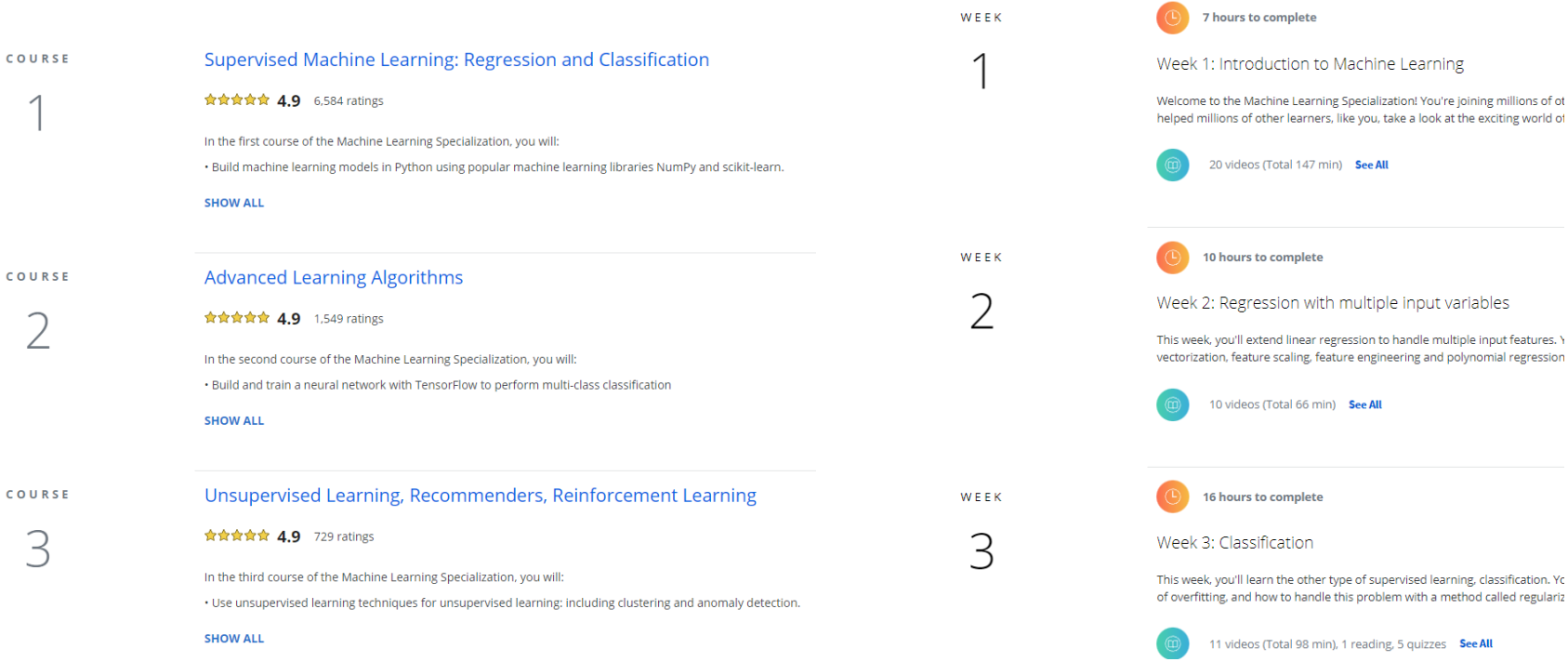






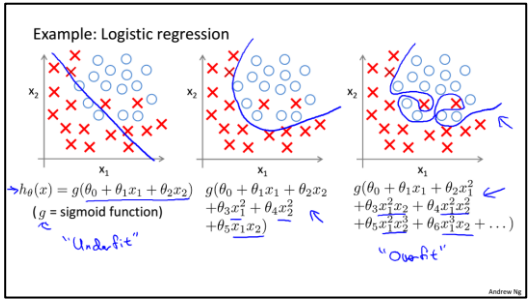
Core ML/DL concepts

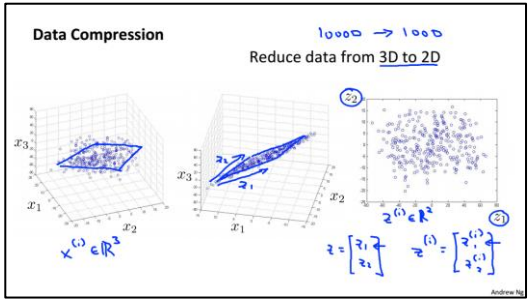
- Coursera: Machine learning by Andrew Ng

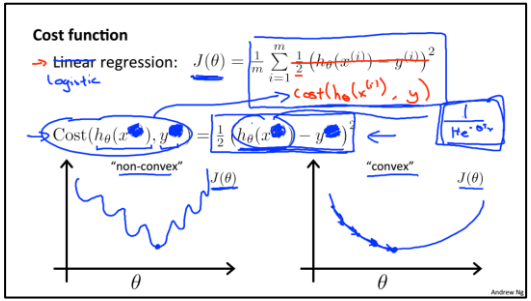


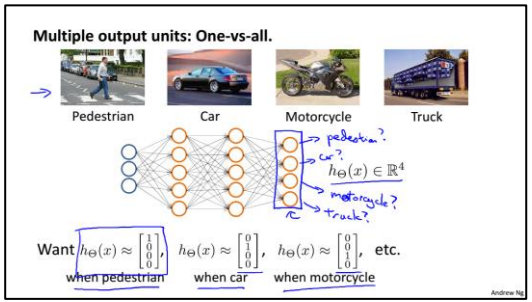
Core ML/DL concepts

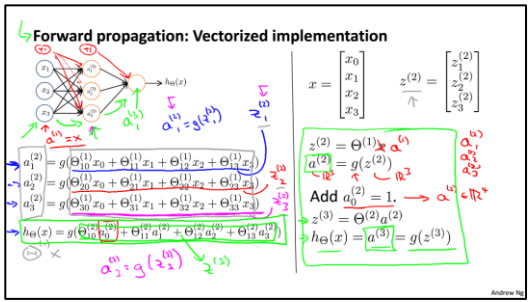
- Coursera: Machine learning by Andrew Ng

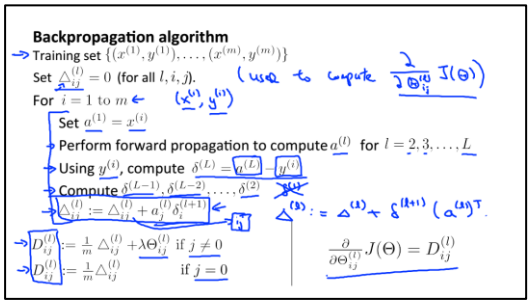






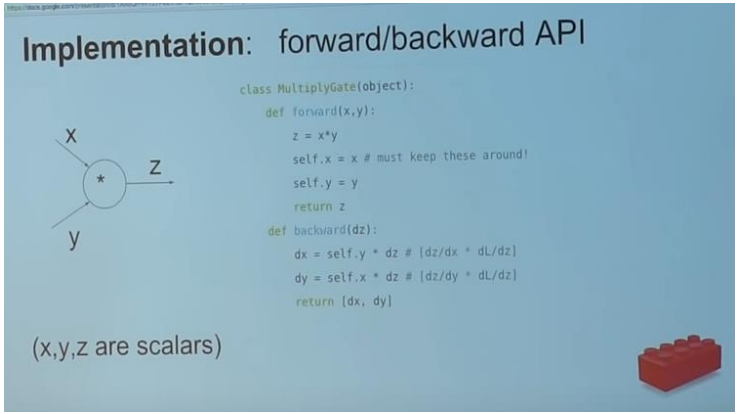


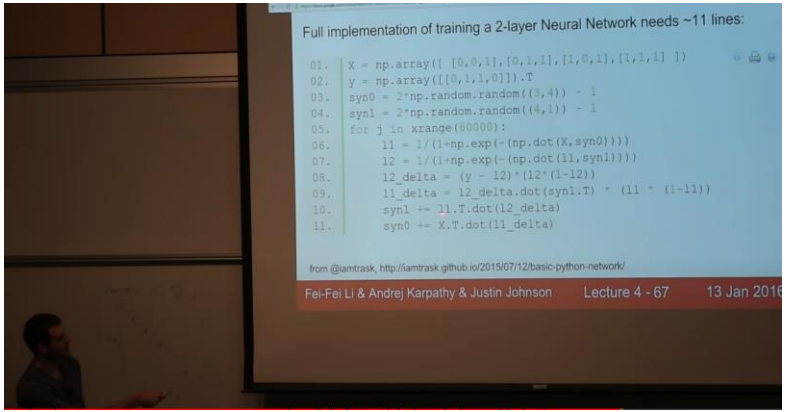


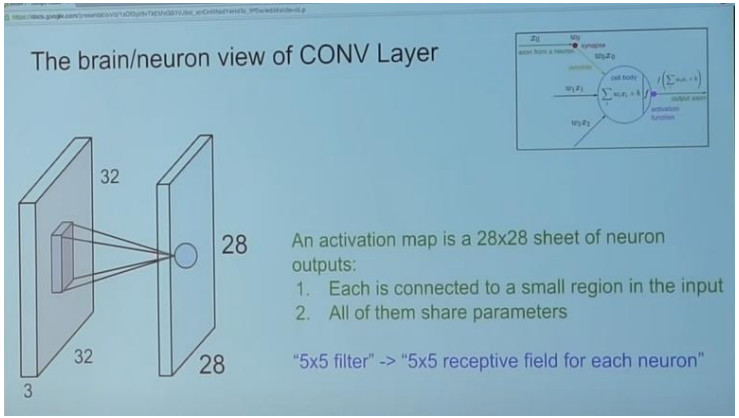


Core ML/DL concepts

- YouTube: cs231n 2016 winter by Andrej Karpathy

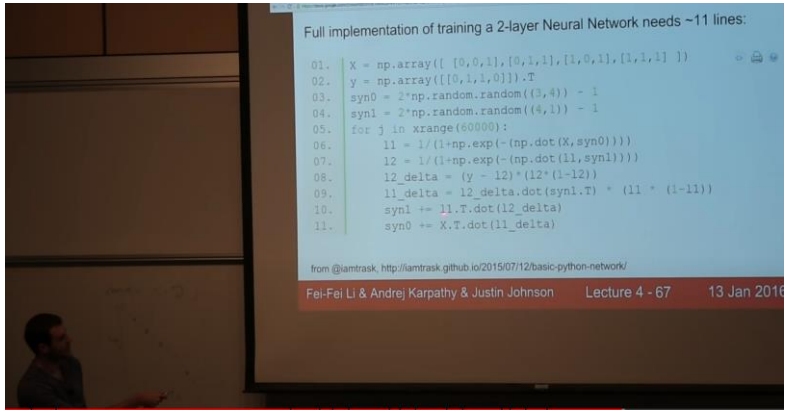






Core ML/DL concepts

- YouTube: cs231n 2016 winter by Andrej Karpathy



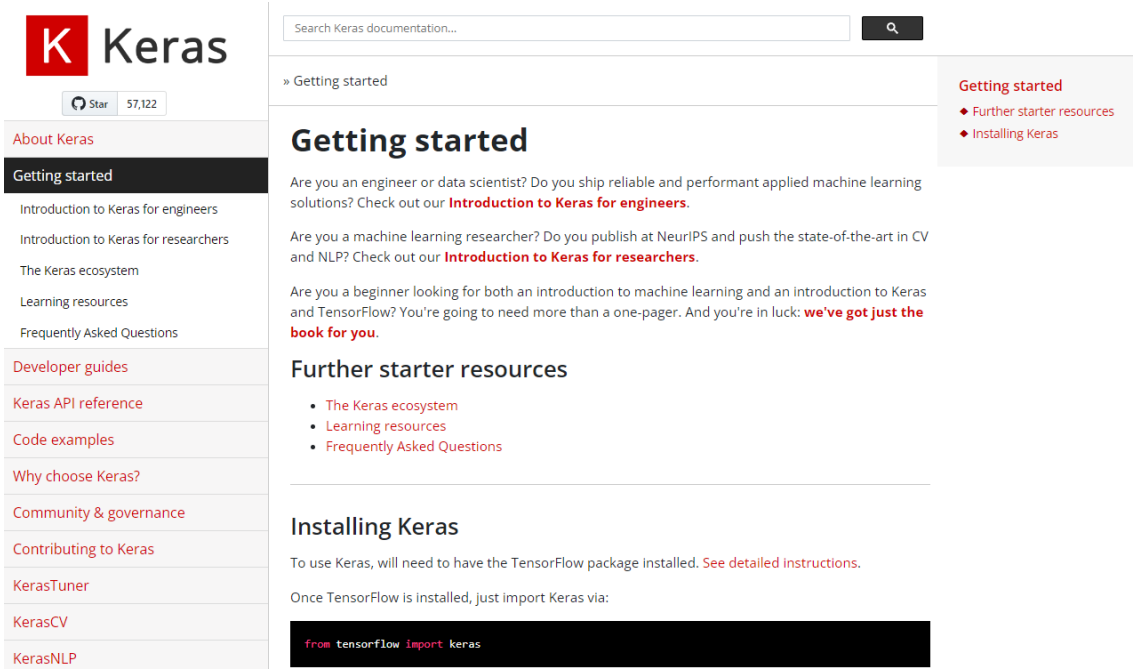

```python
import tensorflow as tf

input_layer = tf.keras.layers.Input(shape=(28,28))
hidden_layer = tf.keras.layers.Dense(units=32, activation='relu')(input_layer)
output_layer = tf.keras.layers.Dense(units=2, activation='softmax')(hidden_layer)

model = tf.keras.Model(inputs=input_layer, outputs=output_layer)
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(x_train, y_train, epochs=10, batch_size=4)
```

Tools and frameworks

- Pytorch
- Tensorflow
- Keras

- Pandas
- NumPy
- Matplotlib
- OpenCV
- ROS

- Git/GitLab
- Linux

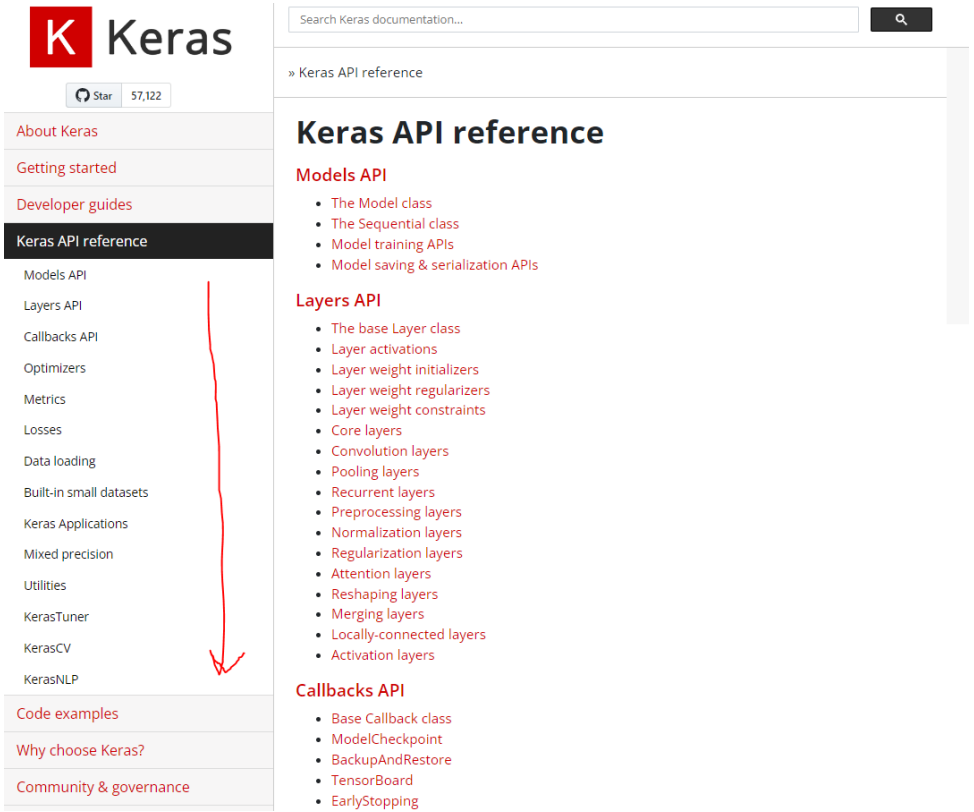


Tools and frameworks

- Pytorch
- Tensorflow
- Keras

- Pandas
- NumPy
- Matplotlib
- OpenCV
- ROS

- Git/GitLab
- Linux

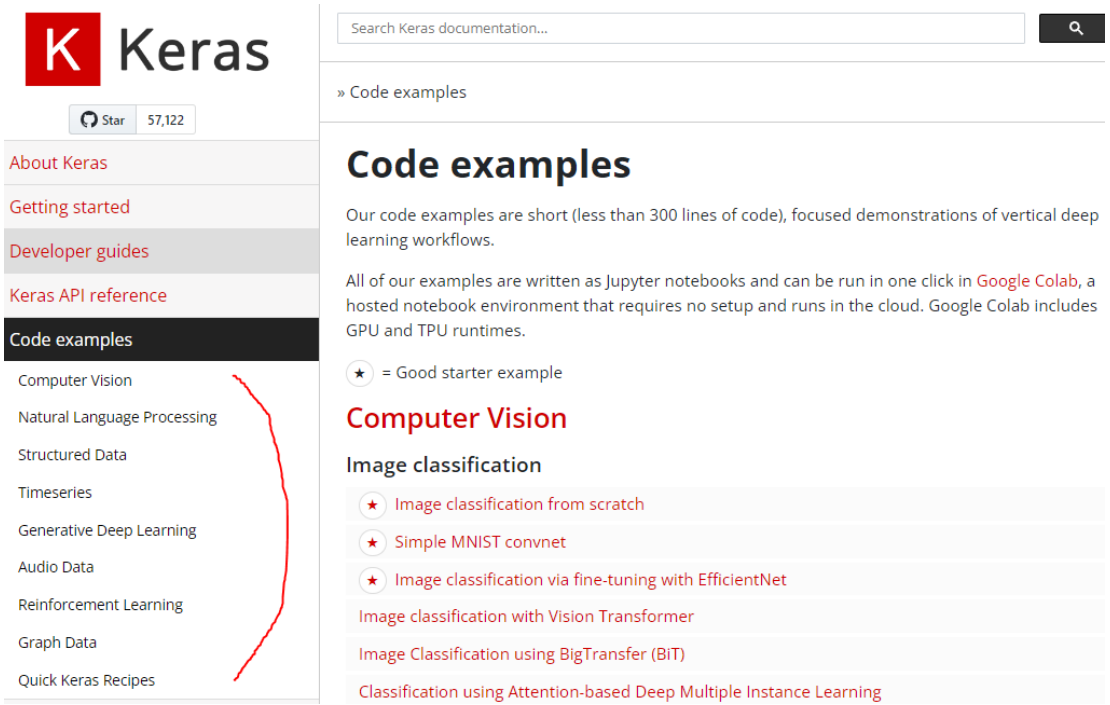


Tools and frameworks

- Pytorch
- Tensorflow
- Keras

- Pandas
- NumPy
- Matplotlib
- OpenCV
- ROS

- Git/GitLab
- Linux

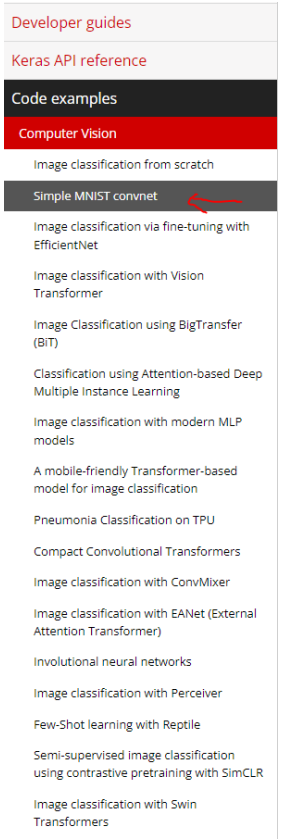


Tools and frameworks

- Pytorch
- Tensorflow
- Keras

- Pandas
- NumPy
- Matplotlib
- OpenCV
- ROS

- Git/GitLab
- Linux



Date created: 2015/06/19
Last modified: 2020/04/21
Description: A simple convnet that achieves ~99% test accuracy on MNIST.

View in Colab · GitHub source

Setup

```
import numpy as np
from tensorflow import keras
from tensorflow.keras import layers
```

Prepare the data

```
# Model / data parameters
num_classes = 10
input_shape = (28, 28, 1)

# Load the data and split it between train and test sets
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

# Scale images to the [0, 1] range
x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255
# Make sure images have shape (28, 28, 1)
x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)
print("x_train shape:", x_train.shape)
print(x_train.shape[0], "train samples")
print(x_test.shape[0], "test samples")


# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

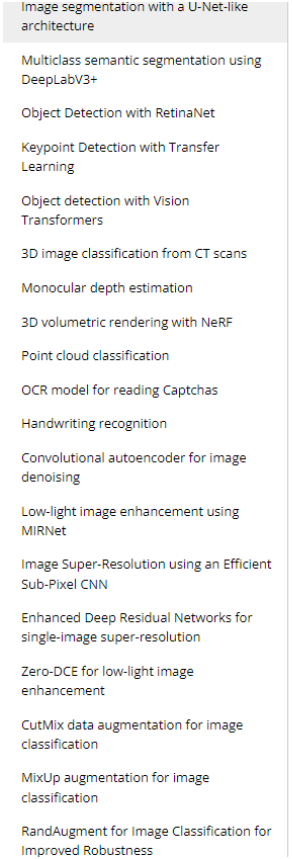
```
x_train shape: (60000, 28, 28, 1)
```

Tools and frameworks

- Pytorch
- Tensorflow
- Keras

- Pandas
- NumPy
- Matplotlib
- OpenCV
- ROS

- Git/GitLab
- Linux



Build the model

```
model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation="softmax"),
    ]
)

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 1600)              0
_________________________________________________________________
dropout (Dropout)            (None, 1600)              0
_________________________________________________________________
dense (Dense)                (None, 10)                16010
=================================================================
Total params: 34,826
Trainable params: 34,826
Non-trainable params: 0
_________________________________________________________________
```

Tools and frameworks

Train the model

```
batch_size = 128
epochs = 15

model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])

model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_split=0.1)
```

```
Epoch 1/15
422/422 [==============================] - 13s 29ms/step - loss: 0.7840 - accuracy: 0.7643 - val_
Epoch 2/15
422/422 [==============================] - 13s 31ms/step - loss: 0.1199 - accuracy: 0.9639 - val_
Epoch 3/15
422/422 [==============================] - 14s 33ms/step - loss: 0.0845 - accuracy: 0.9737 - val_
Epoch 4/15
422/422 [==============================] - 14s 33ms/step - loss: 0.0762 - accuracy: 0.9756 - val_
Epoch 5/15
422/422 [==============================] - 15s 35ms/step - loss: 0.0621 - accuracy: 0.9812 - val_
Epoch 6/15
422/422 [==============================] - 17s 40ms/step - loss: 0.0547 - accuracy: 0.9825 - val_
Epoch 7/15
422/422 [==============================] - 17s 41ms/step - loss: 0.0497 - accuracy: 0.9840 - val_
Epoch 8/15
422/422 [==============================] - 16s 39ms/step - loss: 0.0443 - accuracy: 0.9862 - val_
Epoch 9/15
422/422 [==============================] - 17s 39ms/step - loss: 0.0436 - accuracy: 0.9860 - val_
Epoch 10/15
422/422 [==============================] - 16s 38ms/step - loss: 0.0407 - accuracy: 0.9865 - val_
Epoch 11/15
422/422 [==============================] - 16s 37ms/step - loss: 0.0406 - accuracy: 0.9874 - val_
Epoch 12/15
237/422 [==============>.............] - ETA: 7s - loss: 0.0398 - accuracy: 0.9877
```

Available optimizers

- SGD
- RMSprop
- Adam
- AdamW
- Adadelta
- Adagrad
- Adamax
- Adafactor
- Nadam
- Ftrl

Tools and frameworks

- Pytorch
- Tensorflow
- Keras

- Pandas
- NumPy
- Matplotlib
- OpenCV
- ROS

- Git/GitLab
- Linux

Evaluate the trained model

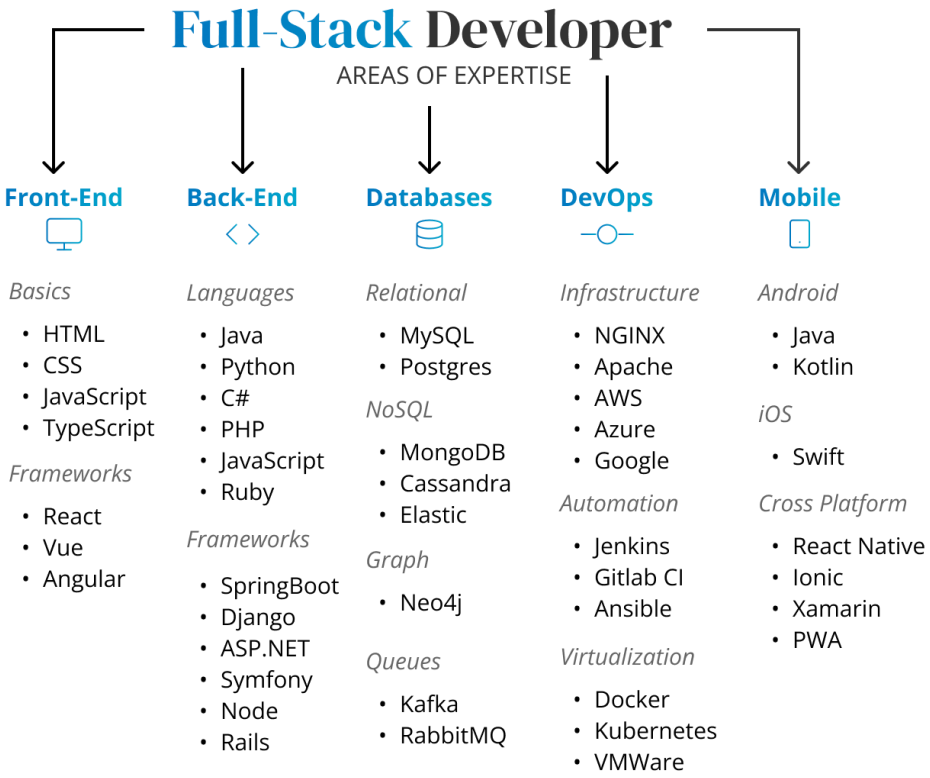
```
score = model.evaluate(x_test, y_test, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

```
Test loss: 0.023950600996613503
Test accuracy: 0.9922000169754028
```

Different levels of engineers

- **Junior engineer/developer**: great coder, knows fundamentals, returns what is asked
- **Medior/middle**: junior + good understanding of the software development process, and is able to work independently
- **Senior**: Middle + birds-eye view skills, deeper technical skillset, knows about team roles and processes, system design expert
- **Lead**: Senior + management

Different levels of engineers



Different levels of engineers

Full-Stack Developer

AREAS OF EXPERTISE

Front-End

Basics
- HTML
- CSS
- JavaScript
- TypeScript

Frameworks
- React
- Vue
- Angular

Back-End

Languages
- Java
- Python
- C#
- PHP
- JavaScript
- Ruby

Frameworks
- SpringBoot
- Django
- ASP.NET
- Symfony
- Node
- Rails

Databases

Relational
- MySQL
- Postgres

NoSQL
- MongoDB
- Cassandra
- Elastic

Graph
- Neo4j

Queues
- Kafka
- RabbitMQ

DevOps

Infrastructure
- NGINX
- Apache
- AWS
- Azure
- Google

Automation
- Jenkins
- Gitlab CI
- Ansible

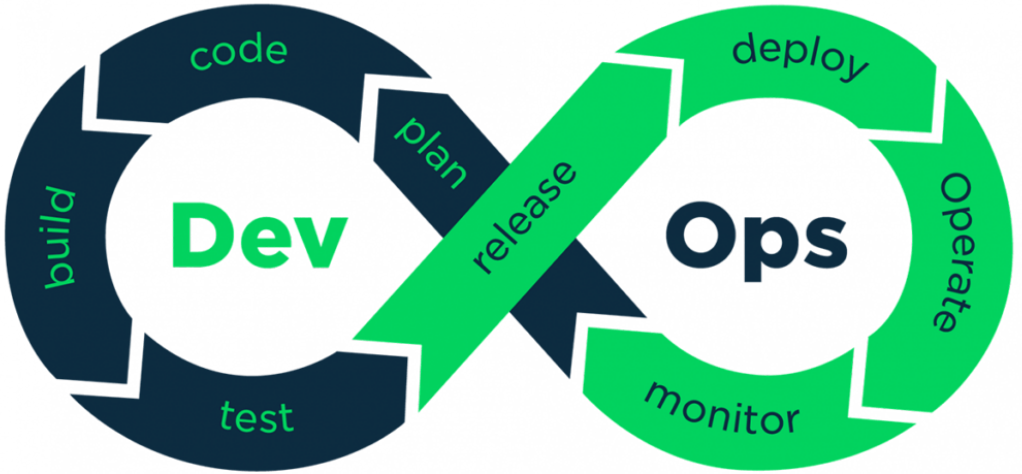
Virtualization
- Docker
- Kubernetes
- VMWare

Mobile

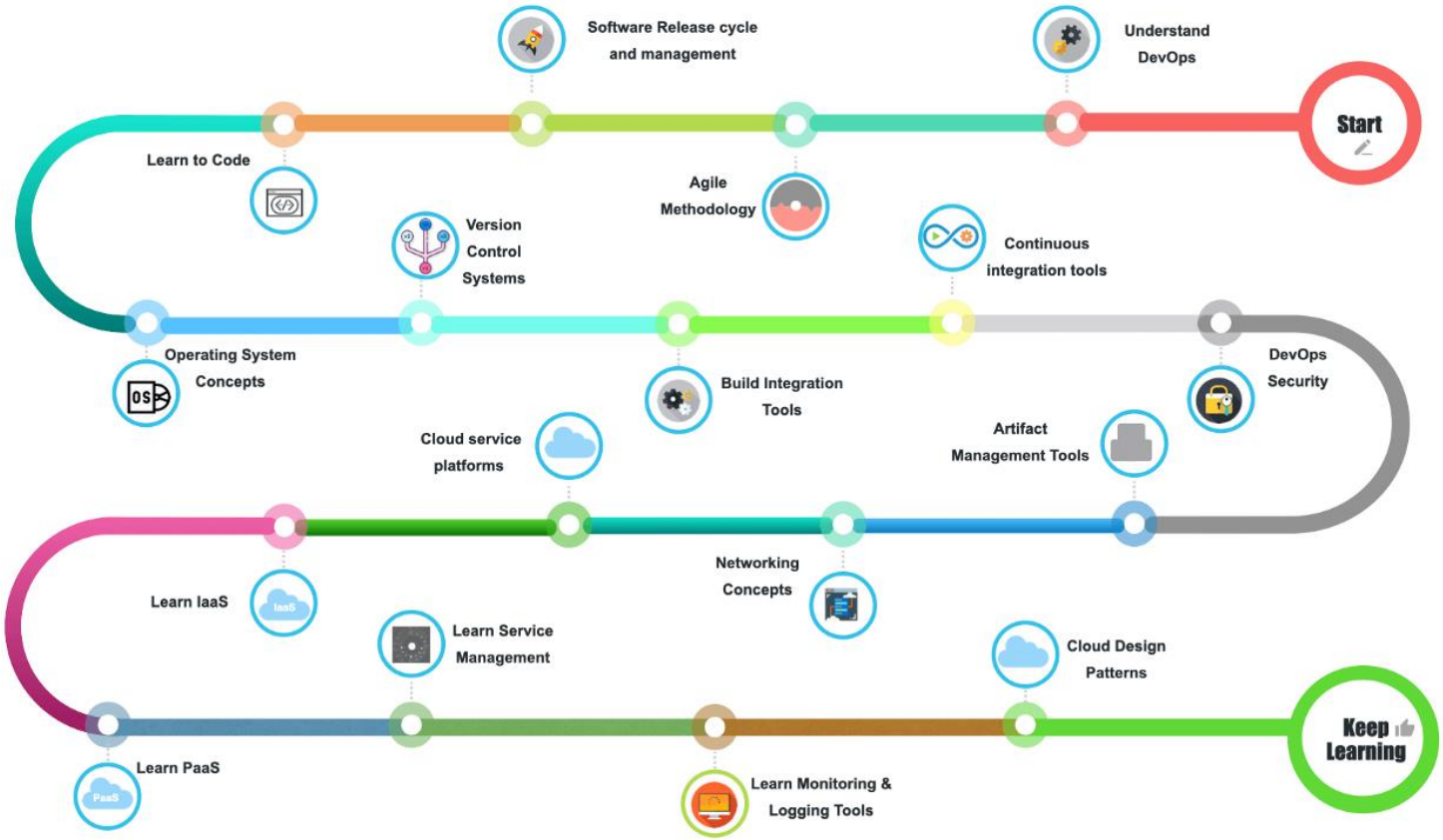
Android
- Java
- Kotlin

iOS
- Swift

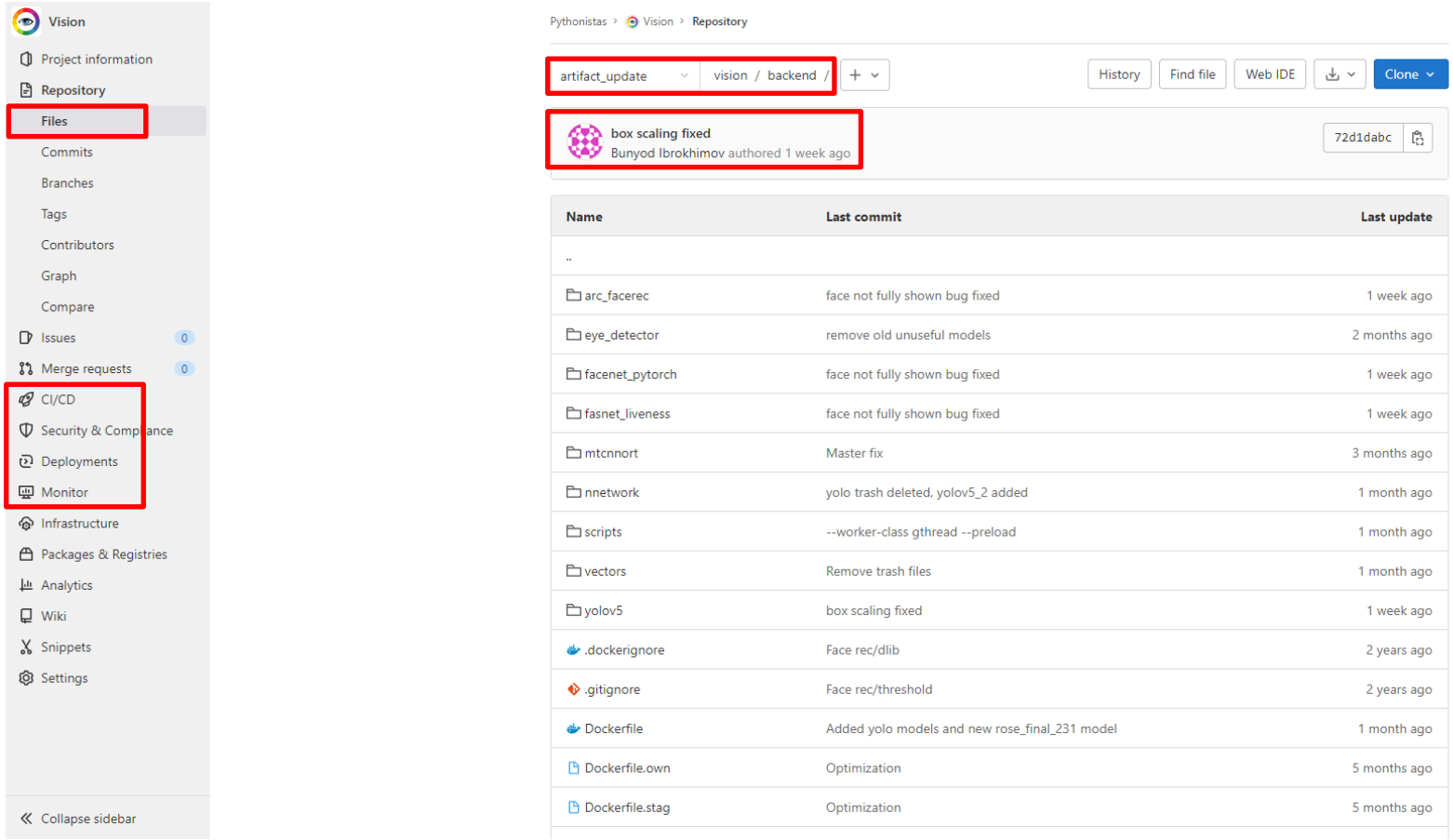
Cross Platform
- React Native
- Ionic
- Xamarin
- PWA



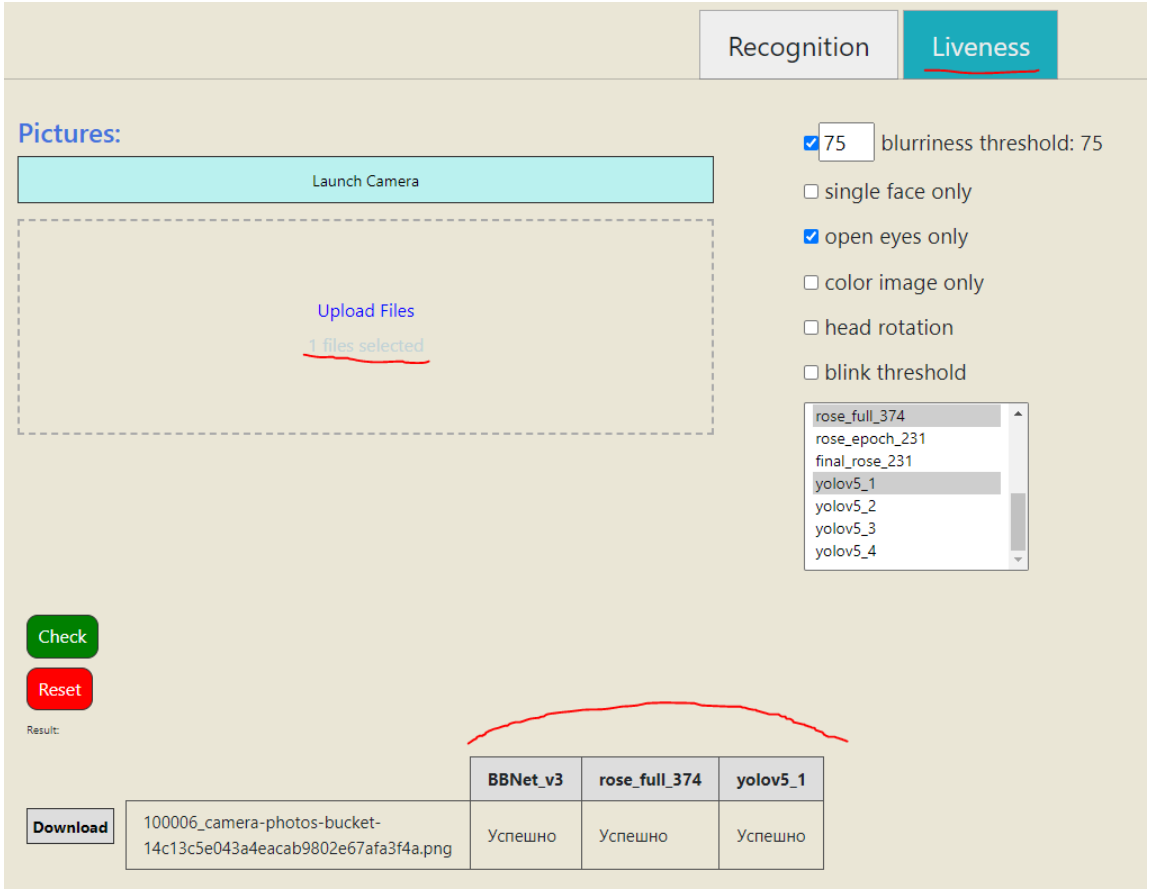
Different levels of engineers



Different levels of engineers



Different levels of engineers



Tips to become a pro

Road to become a senior engineer:

- Write well-design code (clean code: maximize cohesion, minimize coupling)
- Learn dependency injection and abstraction
- Document what you learn
- Always have side projects

Tips to become a pro

- OOP – object-oriented programming
  - Inheritance, abstraction, protocols
- Functional programming
- Optimization (make working codes more efficient, e.g., space, time)
- Modular programming
- SOLID principles
- Test-driven development

Tips to become a pro

Road to become a senior engineer:

- Write well-design code (clean code: maximize cohesion, minimize coupling)
- Learn dependency injection and abstraction
- Document what you learn
- Always have side projects

SOLID principle (Uncle Bob principle)

SOLID

From Wikipedia, the free encyclopedia

> *This article is about the SOLID principles of object-oriented programming. For the fundamental state of matter, see Solid. For other uses, see Solid (disambiguation).*

In software engineering, **SOLID** is a mnemonic acronym for five design principles intended to make object-oriented designs more understandable, flexible, and maintainable. The principles are a subset of many principles promoted by American software engineer and instructor Robert C. Martin,[1][2][3] first introduced in his 2000 paper *Design Principles and Design Patterns* discussing software rot.[2][4]:2–3

SOLID

Principles

Single responsibility
Open–closed
Liskov substitution
Interface segregation
Dependency inversion

V · T · E

The SOLID ideas are

- The **S**ingle-responsibility principle: "There should never be more than one reason for a class to change."[5] In other words, every class should have only one responsibility.[6]
- The **O**pen–closed principle: "Software entities ... should be open for extension, but closed for modification."[7]
- The **L**iskov substitution principle: "Functions that use pointers or references to base classes must be able to use objects of derived classes without knowing it."[8] See also design by contract.[8]
- The **I**nterface segregation principle: "Clients should not be forced to depend upon interfaces that they do not use."[9][4]
- The **D**ependency inversion principle: "Depend upon abstractions, [not] concretions."[10][4]

The SOLID acronym was introduced later, around 2004, by Michael Feathers.[11]

Although the SOLID principles apply to any object-oriented design, they can also form a core philosophy for methodologies such as agile development or adaptive software development.[3]

origin.py > ...

```python
class Order:
    items = []
    quantities = []
    prices = []
    status = 'open'

    def add_item(self, name, quantity, price):
        self.items.append(name)
        self.quantities.append(quantity)
        self.prices.append(price)

    def total_price(self):
        total = 0
        for i in range(len(self.prices)):
            total += self.quantities[i] * self.prices[i]
        return total

    def pay(self, payment_type, security_code):
        if payment_type == 'debit':
            print('Processing debit payment type')
            print(f'Verifying security code: {security_code}')
            self.status = 'paid'
        elif payment_type == 'credit':
            print('Processing credit payment type')
            print(f'Verifying security code: {security_code}')
            self.status = 'paid'
        else:
            raise Exception(f"Unknown payment type: {payment_type}")

order = Order()
order.add_item("iPhone 12 Pro", 1, 1290)
order.add_item("Macbook Air", 1, 1100)

print("Total price: ", order.total_price())
order.pay('debit', '03214')
```

Total price: 2390
Processing debit payment type
Verifying security code: 03214

```
# The Single-responsibility principle:
# There should never be more than one reason
for a class to change.
# In other words, every class should have
only one responsibility.

# In the previous code, Order class has more
than one responsibility, lets fix that.
```

origin.py > ...

```python
class Order:
    items = []
    quantities = []
    prices = []
    status = 'open'

    def add_item(self, name, quantity, price):
        self.items.append(name)
        self.quantities.append(quantity)
        self.prices.append(price)

    def total_price(self):
        total = 0
        for i in range(len(self.prices)):
            total += self.quantities[i] * self.prices[i]
        return total

    def pay(self, payment_type, security_code):
        if payment_type == 'debit':
            print('Processing debit payment type')
            print(f'Verifying security code: {security_code}')
            self.status = 'paid'
        elif payment_type == 'credit':
            print('Processing credit payment type')
            print(f'Verifying security code: {security_code}')
            self.status = 'paid'
        else:
            raise Exception(f"Unknown payment type: {payment_type}")

order = Order()
order.add_item("iPhone 12 Pro", 1, 1290)
order.add_item("Macbook Air", 1, 1100)

print("Total price: ", order.total_price())
order.pay('debit', '03214')
```

```python
class Order:
    items = []
    quantities = []
    prices = []
    status = 'open'

    def add_item(self, name, quantity, price):
        self.items.append(name)
        self.quantities.append(quantity)
        self.prices.append(price)

    def total_price(self):
        total = 0
        for i in range(len(self.prices)):
            total += self.quantities[i] * self.prices[i]
        return total

class PaymentProcessor:
    def debit_pay(self, order, security_code):
        print('Processing debit payment type')
        print(f'Verifying security code: {security_code}')
        order.status = 'paid'

    def credit_pay(self, order, security_code):
        print('Processing credit payment type')
        print(f'Verifying security code: {security_code}')
        order.status = 'paid'


order = Order()
order.add_item("iPhone 12 Pro", 1, 1290)
order.add_item("Macbook Air", 1, 1100)

payment = PaymentProcessor()
print("Total price: ", order.total_price())
payment.credit_pay(order, '03214')
```

```python
class Order:
    items = []
    quantities = []
    prices = []
    status = 'open'

    def add_item(self, name, quantity, price):
        self.items.append(name)
        self.quantities.append(quantity)
        self.prices.append(price)

    def total_price(self):
        total = 0
        for i in range(len(self.prices)):
            total += self.quantities[i] * self.prices[i]
        return total

class PaymentProcessor:
    def debit_pay(self, order, security_code):
        print('Processing debit payment type')
        print(f'Verifying security code: {security_code}')
        order.status = 'paid'

    def credit_pay(self, order, security_code):
        print('Processing credit payment type')
        print(f'Verifying security code: {security_code}')
        order.status = 'paid'


order = Order()
order.add_item("iPhone 12 Pro", 1, 1290)
order.add_item("Macbook Air", 1, 1100)

payment = PaymentProcessor()
print("Total price: ", order.total_price())
payment.credit_pay(order, '03214')

```

```
# The Open-closed principle
# Software entities ... should be open for extension
but closed for modification.

# In the previous code we cannot add new payment
methods without changing the class. Let's fix
that...
```

```python
class Order:
    items = []
    quantities = []
    prices = []
    status = 'open'

    def add_item(self, name, quantity, price):
        self.items.append(name)
        self.quantities.append(quantity)
        self.prices.append(price)

    def total_price(self):
        total = 0
        for i in range(len(self.prices)):
            total += self.quantities[i] * self.prices[i]
        return total

class PaymentProcessor:
    def debit_pay(self, order, security_code):
        print('Processing debit payment type')
        print(f'Verifying security code: {security_code}')
        order.status = 'paid'

    def credit_pay(self, order, security_code):
        print('Processing credit payment type')
        print(f'Verifying security code: {security_code}')
        order.status = 'paid'


order = Order()
order.add_item("iPhone 12 Pro", 1, 1290)
order.add_item("Macbook Air", 1, 1100)

payment = PaymentProcessor()
print("Total price: ", order.total_price())
payment.credit_pay(order, '03214')

```

```python
from abc import ABC, abstractmethod
```

```python
class PaymentProcessor(ABC):
    @abstractmethod
    def pay(self, order, security_code):
        pass

class DebitPaymentProcessor:
    def pay(self, order, security_code):
        print('Processing debit payment type')
        print(f'Verifying security code: {security_code}')
        order.status = 'paid'

class CreditPaymentProcessor:
    def pay(self, order, security_code):
        print('Processing credit payment type')
        print(f'Verifying security code: {security_code}')
        order.status = 'paid'

class PaypalPaymentProcessor:
    def pay(self, order, security_code):
        print('Processing paypal payment type')
        print(f'Verifying security code: {security_code}')
        order.status = 'paid'

order = Order()
order.add_item("iPhone 12 Pro", 1, 1290)
order.add_item("Macbook Air", 1, 1100)

payment = PaypalPaymentProcessor()
print("Total price:", order.total_price())
payment.pay(order, '03214')
```

```python
from abc import ABC, abstractmethod

class PaymentProcessor(ABC):
    @abstractmethod
    def pay(self, order, security_code):
        pass

class DebitPaymentProcessor:
    def pay(self, order, security_code):
        print('Processing debit payment type')
        print(f'Verifying security code: {security_code}')
        order.status = 'paid'

class CreditPaymentProcessor:
    def pay(self, order, security_code):
        print('Processing credit payment type')
        print(f'Verifying security code: {security_code}')
        order.status = 'paid'

class PaypalPaymentProcessor:
    def pay(self, order, security_code):
        print('Processing paypal payment type')
        print(f'Verifying security code: {security_code}')
        order.status = 'paid'

order = Order()
order.add_item("iPhone 12 Pro", 1, 1290)
order.add_item("Macbook Air", 1, 1100)

payment = PaypalPaymentProcessor()
print("Total price:", order.total_price())
payment.pay(order, '03214')
```

```
# The Liskov substitution principle:
# Functions that use pointers or references to base
classes must be able to use objects of derived
classes without knowing it.

# In the previous code, if we want to change
security_code to email_address for paypal payment
method, we are abusing PaymentProcessor class for
something it is not supposed to do. And this
violates Liskov substitution principle. Let's fix
that...
```

```python
from abc import ABC, abstractmethod
```

```python
class PaymentProcessor(ABC):
    @abstractmethod
    def pay(self, order, security_code):
        pass

class DebitPaymentProcessor:
    def pay(self, order, security_code):
        print('Processing debit payment type')
        print(f'Verifying security code: {security_code}')
        order.status = 'paid'

class CreditPaymentProcessor:
    def pay(self, order, security_code):
        print('Processing credit payment type')
        print(f'Verifying security code: {security_code}')
        order.status = 'paid'

class PaypalPaymentProcessor:
    def pay(self, order, security_code):
        print('Processing paypal payment type')
        print(f'Verifying security code: {security_code}')
        order.status = 'paid'

order = Order()
order.add_item("iPhone 12 Pro", 1, 1290)
order.add_item("Macbook Air", 1, 1100)

payment = PaypalPaymentProcessor()
print("Total price:", order.total_price())
payment.pay(order, '03214')
```

```python
class DebitPaymentProcessor(PaymentProcessor):
    def __init__(self, security_code):
        self.security_code = security_code

    def pay(self, order):
        print('Processing debit payment type')
        print(f'Verifying security code: {self.security_code}')
        order.status = 'paid'

class CreditPaymentProcessor(PaymentProcessor):
    def __init__(self, security_code):
        self.security_code = security_code

    def pay(self, order):
        print('Processing credit payment type')
        print(f'Verifying security code: {self.security_code}')
        order.status = 'paid'

class PaypalPaymentProcessor(PaymentProcessor):
    def __init__(self, email_address):
        self.email_address = email_address

    def pay(self, order):
        print('Processing paypal payment type')
        print(f'Verifying email address: {self.email_address}')
        order.status = 'paid'

order = Order()
order.add_item("iPhone 12 Pro", 1, 1290)
order.add_item("Macbook Air", 1, 1100)

payment = CreditPaymentProcessor("04132")
print("Total price:", order.total_price())
payment.pay(order)
```

Homework: finish I and D

```
# The Interface segregation principle:
# Clients should not be forced to depend
upon interfaces that they do not use.

# In the previous code, we are abusing
credit card method to use sms
authenticate,
# And this violates Interface segregation
principle. Let's fix that...
```

```
# The Dependency inversion principle:
# Depend upon abstractions, [not] concretions.

# In the previous code, if we want to add new
authentication method, we have to change SMSAuth
class because it has dependency
# Instead we can create new authorizer class so
that we can easily add new auth types in future.
Let's fix that...
```

Agenda

01

What is software engineering?

02

Where to start:
A Roadmap

03

Different levels of software engineer

04

Tips to become a pro

Thank You

Q & A