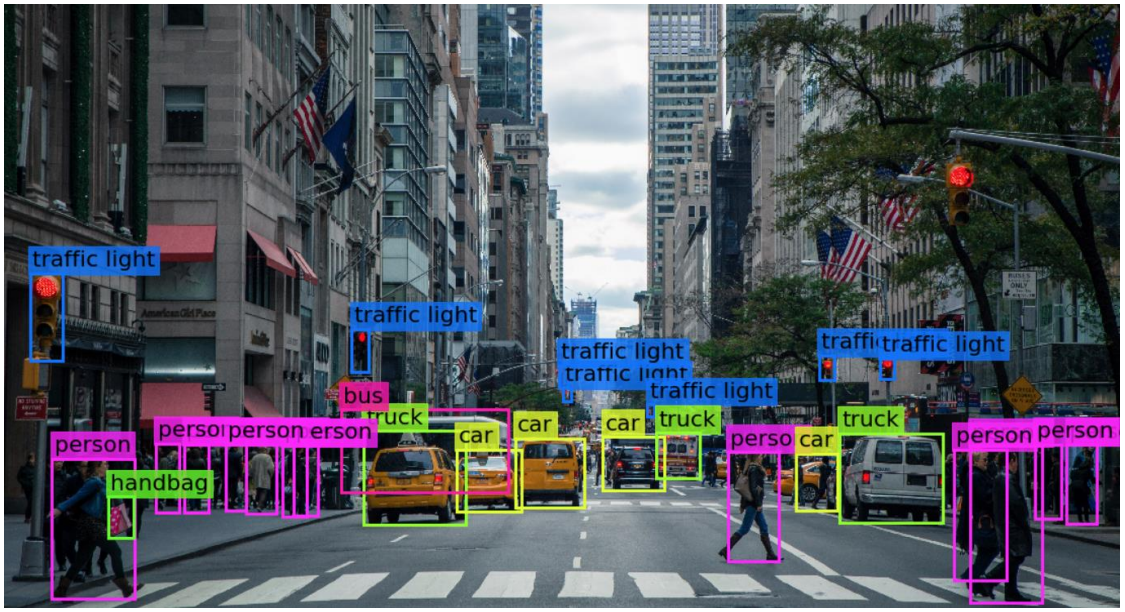


Computer vision in the new era of Artificial Intelligence and Deep Learning

Visión por computador en la nueva era de la Inteligencia Artificial y el Deep Learning

Rubén Usamentiaga*, Alberto Fernández°
*** University of Oviedo**
° TSK

Gijón (Spain)
5 – 16 April 2021

https://github.com/albertofernandezvillan/computer-vision-and-deep-learning-course

Python

python_introduction.ipynb

python_introduction.ipynb

https://github.com/albertofernandezvillan/computer-vision-and-deep-learning-course

Python introduction

- Python is an interpreted, high-level and general-purpose programming language
- Python can be used for scientific computing:
  - Libraries such as NumPy, SciPy and Matplotlib allow the effective use of Python in scientific computing
  - OpenCV has python bindings with a rich set of features for computer vision and image processing
  - Python is commonly used in artificial intelligence projects and machine learning projects with the help of libraries like TensorFlow, Keras, Pytorch and Scikit-learn

Programming examples

Python is a high-level, dynamically typed multiparadigm programming language. As an example, see a Python function to calculate the factorial of a number (a non-negative integer)

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)

n=int(input("Input a number to compute the factorial : "))

print(factorial(n))
```

Variables

Variables are containers for holding data and they're defined by a name and value

```
# This is a integer variable
x = 10
```

```
# This is a float variable
x = 10.0
```

```
# This is a string variable
x = '10.0'
```

```
# This is a boolean variable
x = True
```

We can also do operations with variables

```
x = 1
y = 2
z = (x + y) / 2
```

```
a = "this is "
b = "an example"
c = a + b
```

```
x = 4

print(x + 1)     #  Addition
print(x - 1)     #  Subtraction
print(x * 2)     #  Multiplication
print(x ** 2)    #  Exponentiation
print(x % 3)     #  Modulo
```

Importing modules

In python, we can import modules, which provide specific functions. For example, we can import math module, which provides access to the mathematical function

```
# Import the math module
import math

x = 4
print(math.sqrt(4))  # Square root
```

```
# Import only the method to be used
from math import factorial

print(factorial(4))  # Factorial
```

Data types

Python provides 4 built-in data types in used to store **collections of data**

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

Data types to store collections of data

List Vs Set Vs Dictionary Vs Tuple

Lists	Sets	Dictionaries	Tuples
List = [10, 12, 15]	Set = {1, 23, 34} Print(set) -> {1, 23,24} Set = {1, 1} print(set) -> {1}	Dict = {"Ram": 26, "mary": 24}	Words = ("spam", "egss") Or Words = "spam", "eggs"
Access: print(list[0])	Print(set). Set elements can't be indexed.	print(dict["ram"])	Print(words[0])
Can contains duplicate elements	Can't contain duplicate elements. Faster compared to Lists	Can't contain duplicate keys, but can contain duplicate values	Can contains duplicate elements. Faster compared to Lists
List[0] = 100	set.add(7)	Dict["Ram"] = 27	Words[0] = "care" ->TypeError
Mutable	Mutable	Mutable	Immutable - Values can't be changed once assigned
List = []	Set = set()	Dict = {}	Words = ()
Slicing can be done print(list[1:2]) -> [12]	Slicing: Not done.	Slicing: Not done	Slicing can also be done on tuples
Usage: Use lists if you have a collection of data that doesn't need random access. Use lists when you need a simple, iterable collection that is modified frequently.	Usage: - Membership testing and the elimination of duplicate entries. - when you need uniqueness for the elements.	Usage: - When you need a logical association b/w key:value pair. - when you need fast lookup for your data, based on a custom key. - when your data is being constantly modified.	Usage: Use tuples when your data cannot change. A tuple is used in comibnation with a dictionary, for example, a tuple might represent a key, because its immutable.

Data types

`<class 'list'>`

```
# Creating a list:
my_list = [1,2,3,4,5]

# Creating empty lists:
my_empty_list = []
my_empty_list_2 = list()
```

`<class 'tuple'>`

```
# Creating a tuple:
my_tuple = (1,2,3,2,2)

# Creating empty tuples:
my_empty_tuple = ()
my_empty_tuple_2 = tuple()
```

`<class 'dict'>`

```
# Creating a dictionary:
my_dict = {'key_1': 10,
           'key_2': 20}

# Creating empty dicts
my_empty_dict = {}
my_empty_dict_2 = dict()
```

`<class 'set'>`

```
# Creating a set:
my_set = {1,2,3,4,5,6}

# Creating an empty set:
my_empty_set = set()
```

Use type(my_object) to get the class type (e.g. type(my_list))

List Comprehensions

List comprehensions provide a concise way to create lists

```
# Calculates the squares
# of 1, 2, ... 9
squares = []
for x in range(10):
  squares.append(x**2)
```

```
squares = [x**2 for x in range(10)]
```

```
# Calculates the squares
# of 1, 3, ... 9
squares_odd = []
for x in range(10):
  if x % 2:
    squares_odd.append(x**2)
```

```
squares_od = [x**2 for x in range(10) if x % 2 ]
```

Creating dictionaries

Dictionaries are used to store data values in key:value pairs

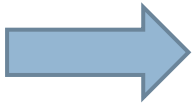
```
# This creates a dictionary:
my_dict = {'key_1': 10, 'key_2': 20}
```

The dict() constructor builds dictionaries directly from sequences of key-value pairs:

```
list_pairs = [("a", 1),("b", 2),("c", 3)]
my_dict = dict(list_pairs)
```

In connection with this previous example, a common way of creating dictionaries in Python is by using the zip() method

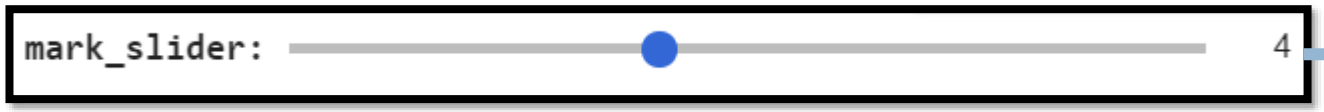
```
letters = ['a', 'b']
nums = [0, 1]
my_dict = dict(zip(letters, nums))
```



```
{'a': 0, 'b': 1}
```

If statements

We can use if statements to conditionally do something. The conditions are defined by the words if, elif and else




```
if mark_slider > 8:
  print("Your mark is: {}".format(mark_slider))
  print("Congratulations you have a high mark!")
elif mark_slider >= 5:
  print("Your mark is: {}".format(mark_slider))
  print("Congratulations you have passed the exam!")
else:
  print("Your mark is: {}".format(mark_slider))
  print("Better luck next time. Keep trying!")
```

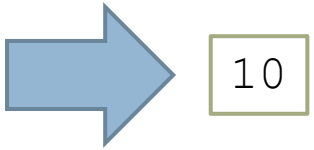
Loops

Python provides following types of loops to handle looping requirements

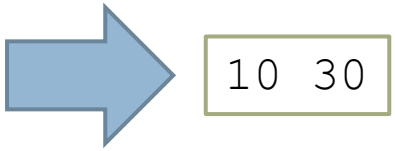
For loops

```
numbers = [10, 20, 30]
for number in numbers:
    print(number)
```


```
numbers = [10, 20, 30]
for number in numbers:
  if number == 20:
    break
  print(number)
```




```
numbers = [10, 20, 30]
for number in numbers:
  if number == 20:
    continue
  print(number)
```



Loops

While loops: A while loop can perform repeatedly as long as a condition is True. We can use continue and break commands in while loops as well

```
x = 10
times = 0

while True:
  if x > 15:
    break
  print("Value of x is: {}".format(x))
  x = x + 2
  times = times + 1

print("Times = {}".format(times))
```

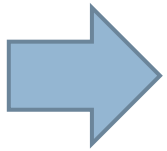
```
Value of x is: 10 Value of x is:
12 Value of x is: 14 Times = 3
```

Functions

Functions are a way to modularize reusable pieces of code. They are defined by the keyword def

```
def my_function():
  print("Hello from a function")

my_function()
```

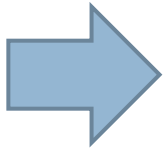


```
Hello from a function
```

```
def my_function(food):
  for x in food:
    print(x)

fruits = ["apple", "banana", "cherry"]

my_function(fruits)
```



```
apple
banana
cherry
```

Classes

Classes provide a means of bundling data and functionality together. Creating a new class creates a new type of object, allowing new instances of that type to be made

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

    def speak(self):
        print("I am", self.name,
              "and I am", self.age, "years old")

    def change_age(self, age):
        self.age = age
```

```
tim = Dog("Tim", 5)
tim.change_age(7)
tim.speak()
```

```
I am Tim and I am 7 years old
```

Python

Recommended lectures

- The Python Tutorial:
  - https://docs.python.org/3/tutorial/
- Learn the Basics:
  - https://www.learnpython.org/en/
- (Spanish: Aprenda las bases):
  - https://www.learnpython.org/es/

https://github.com/albertofernandezvillan/computer-vision-and-deep-learning-course