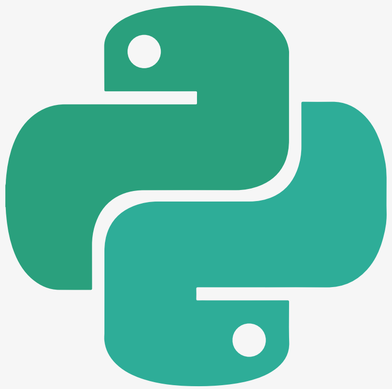


LANGAGE DE PROGRAMMATION

PYTHON

Brad Niepceron - niepceron.brad@gmail.com

CONTENU DU MODULE

TEMPS

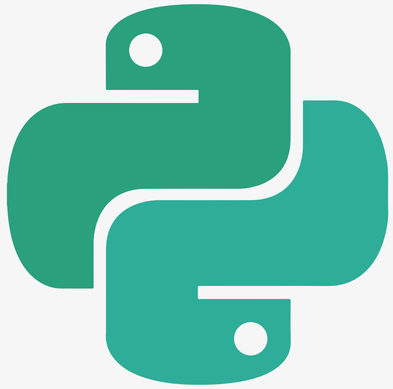
- 12 heures.
  Pause pendant TP.

OBJECTIFS

- Automatisation de tâche
- Anticiper le module multitâche

NOTATION

- Questionnaire + Exo
- Éventuel bonus TP



```python
l1 = [2, 3, 4, 5, 6]
l2 = [[x, x**2] for x in l1[:-1]]
print(l2)
```

PARLEZ-VOUS PYTHON ?

INTRODUCTION À PYTHON

01

01

INTRODUCTION À PYTHON

En gros

- Langage interprété & **« haut niveau »**
- Créé en 1991 par Guido Van Rossum
- Langage **cross-platform**, Windows, Unix, ….
- Langage multi-paradigme
- Typage dynamique **fort**
- Indentation obligatoire délimitant les **scopes**

01 INTRODUCTION À PYTHON

Applicabilité

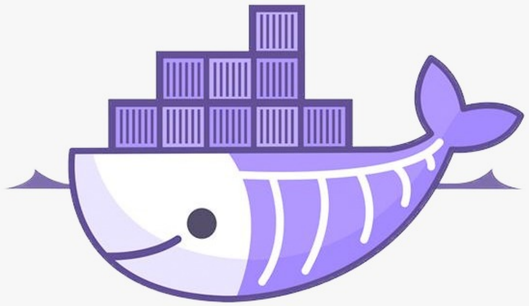
- L’enseignement
- L’administration de système
- Création de logiciel
- L’électronique et l’embarqué (Zerynth)
- Le développement web (Django, Flask -> Instagram, Youtube)
- La science et la recherche (Data science, Machine Learning)

01

INTRODUCTION À PYTHON

Environnement de développement

OÙ



01

INTRODUCTION À PYTHON

Variables

```python
# Variables numériques

int(), float(), complex()
a = 10
b = 10.19
c = (3 + 6j)
type(c)
# > complex
```

```python
# Itérables

list(), tuple(), set(), range()
my_list = [0, 1, 2, 3, 3] # mutable
my_tuple = (0, 1, 2, 3) # immutable
my_set = set(my_list) # [0, 1, 2, 3]
my_range = range(3) # [0, 1, 2]
```

```python
# Autres types

a = None
b = object()
c = True
```

Pas besoin de mentionner le type d'une variable en Python.

01

INTRODUCTION À PYTHON

Opérateurs

```
# Opérateurs

x = 10
y = 3
x - y #7
x + y #13
x * y #30
x / y #3.333333...
x ** y #1000
x % y #1
x // y #3
```

```
x = 10
y = 3
x == y #False
x < y #False
x > y #True
x <= y #False
x >= y #True
x != y #False
```

```
x = True
y = False
z = (True, 22, 'string')
x and y #False
x or y #True
x is True #True
not x #False
x in z #True
```

01

INTRODUCTION À PYTHON

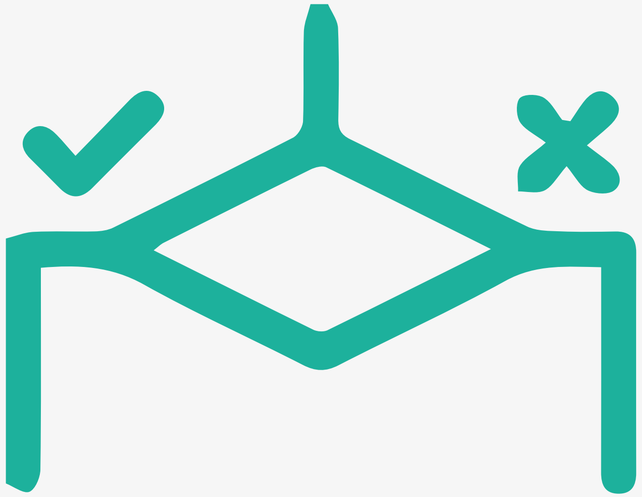
Affichage

```
# Affichage

print('Hello World')
print('Hello {second_name}, my name is {name}'.format(name = 'Brad',
                                                      second_name = 'Toto'))
print('Hello {}, my name is {}'.format('Toto', 'Brad'))
print('Hello '+'Toto \n')
```

01

INTRODUCTION À PYTHON

Condition



- Les keyword **if** et **else** sont utilisés pour la gestion des conditions
- Pour imbriquer des **if**, on utilise **elif**

```
if a > 10:
    pass
    elif a = 5:
        pass
else:
    pass
```

01

INTRODUCTION À PYTHON

Boucle

- Les boucles sont déclarées par **for .. in ..** ou **while ..**
- Une boucle peut être arrêtée par **break**
- Une boucle peut être passée par **continue**

```
for i in [0, 1]:
    print(i)
    break
for i, j in dict():
    continue
while True:
    print(0)
    break
```

01

INTRODUCTION À PYTHON

Générateurs

- Utilise le concept de **lazy evaluation**
- Utile pour l'évaluation de grandes données
- Différent d'un itérateur

```
l, generator = ['toto', 'titi', 'tata', 'tutu'], (x for x in l)
for i in generator:
    print(i)

def data_generator(data):
    for i in data:
        yield i

for i in data_generator([0, 1, 2]):
    print(i)
```

01

INTRODUCTION À PYTHON

Fonction

- Une fonction est déclarée par **def**
- On utilise **return** pour retourner une valeur
- On peut passer des paramètres optionnels

```
def sum(a, b, c=None)
    if c:
        return a + b + c
    else:
        return a + b

sum(5, 10)
```

01

INTRODUCTION À PYTHON

Module & Import

- Tout import se fait avec le keyword **import**
- Des librairies peuvent être ajoutées par un outil comme **pip**
- Un fichier **__init__.py** à la racine d'un projet transforme le projet en module

```
import os
from os import path
```

01

INTRODUCTION À PYTHON

Lecture de fichier

- N’importe quel fichier peut être lu de la manière suivante

```
with open('myfile.txt', 'r') as f:
    # f.readlines()
    # f.write()
```

- Le fichier reste ouvert à l’intérieur de mais doit être fermer si ouvert d’une autre manière

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

01 INTRODUCTION À PYTHON

TP1

Créer une fonction donnant le dictionnaire suivant et prenant un numéro de clé en entrée

```
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64}
```

TP2

Créer une fonction retournant la liste des nombres premiers entre deux bornes

TP3

Créer une fonction prenant 2 entiers **A** et **B** en entrée et créant un liste de **A** sous listes et de **B** valeurs aléatoires. **Tout ça, en une seule ligne.**

TP4

Créer un module contenant les fonctions créées en l'appelant par votre nom, importer le dans un fichier extérieur et appeler les fonctions une à une.

PROGRAMMATION ORIENTÉE OBJET

02

02

PROGRAMMATION ORIENTÉE OBJET

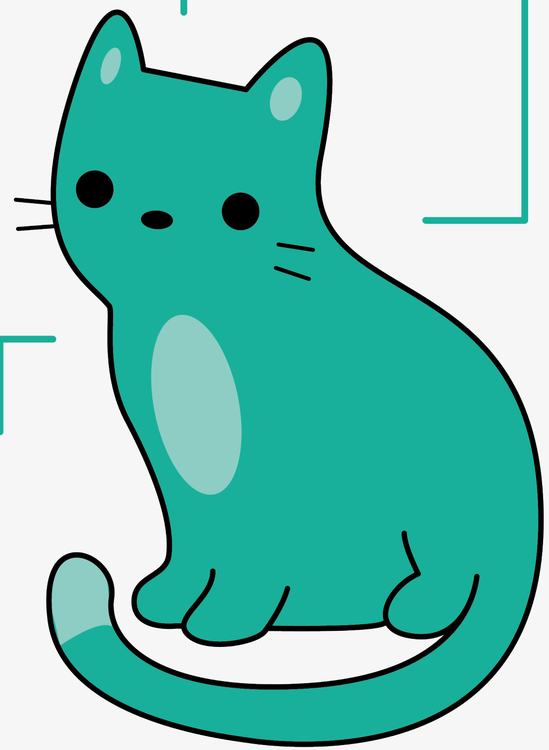
Terminologie

CLASS

Animal

OBJECT

Charlie



PROPERTIES

Name

Colour

METHODS

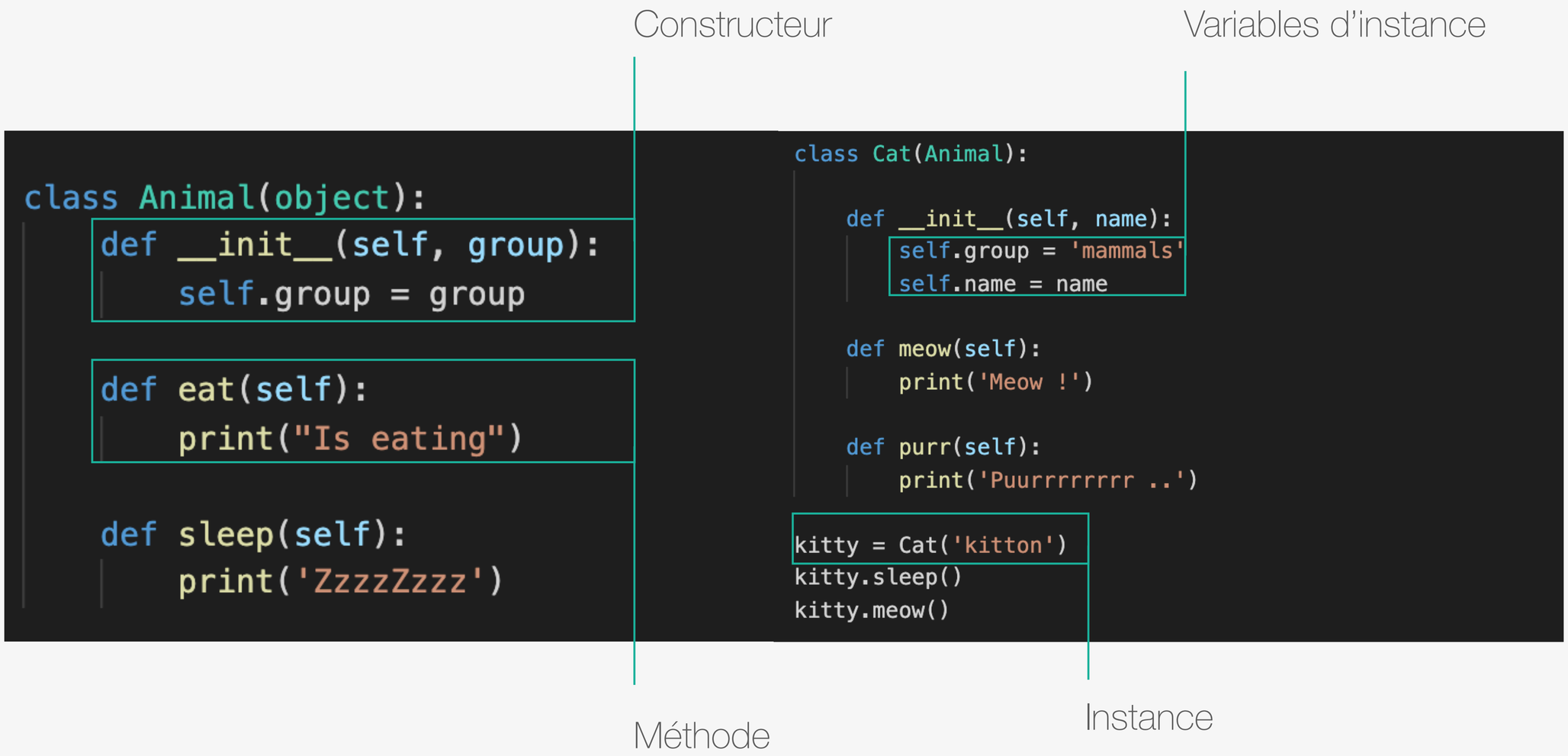
Sleep

Meow

02

PROGRAMMATION ORIENTÉE OBJET

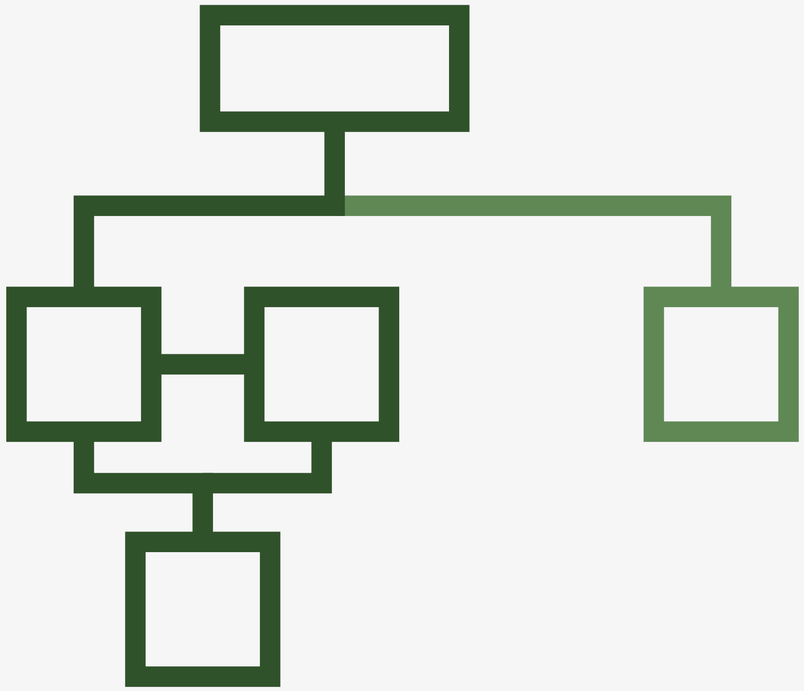
Définition



02

PROGRAMMATION ORIENTÉE OBJET

Héritage multiple



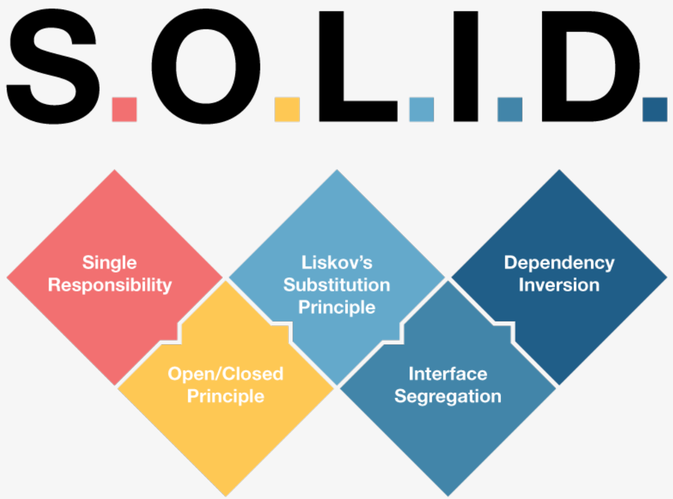
- Une classe peut avoir plusieurs parents
- L'héritage d'attributs et méthodes fonctionne comme pour l'héritage simple
- L'ordre de l'héritage est important pour la récupération d'attributs. On parle de **Method Resolution Order**

```python
class First(object):
    def __init__(self):
        pass

class Second(First):
    def __init__(self):
        pass

class Third(First):
    def __init__(self):
        pass

class Fourth(Second, Third):
    def __init__(self):
        super(Fourth, self).__init__()
        pass
```

02

PROGRAMMATION ORIENTÉE OBJET

Classe abstraite



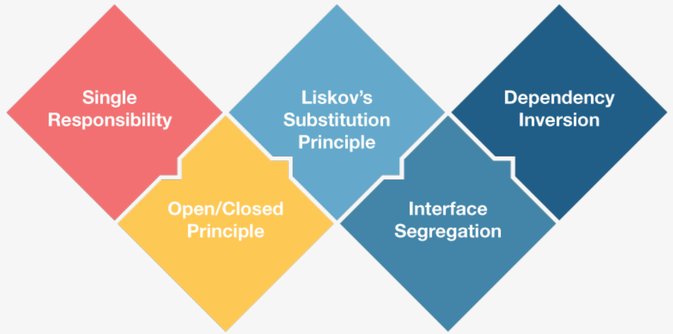
- Permet de créer des interfaces communes pour des implémentations différentes.
- Composée de **méthode abstraite** (sans implémentation)
- Utile pour la création d'**API**
- Bonne pratique pour le respect des principes **S.O.L.I.D**
- Différent d'une **Interface**

02

PROGRAMMATION ORIENTÉE OBJET

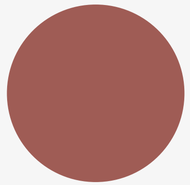
SOLID FOR KNOWLEDGE

S.O.L.I.D.



SINGLE RESPONSIBILITY

Une classe doit avoir une et **une seule** raison de changer.
Elle doit avoir une seule et une seule responsabilité (fonction)



Class **Circle** -> self.radius Class **Square** -> self.length

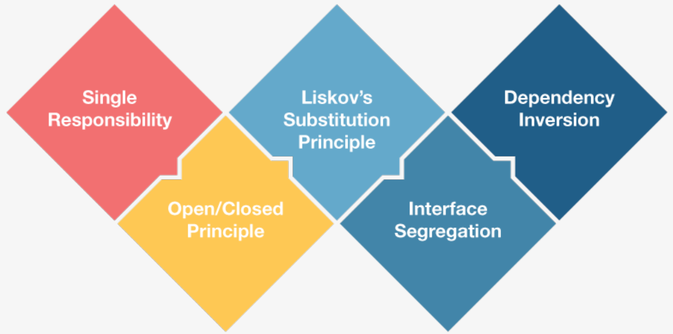
Calculate Area ?

02

PROGRAMMATION ORIENTÉE OBJET

SOLID FOR KNOWLEDGE

S.O.L.I.D.



SINGLE RESPONSIBILITY

Une classe doit avoir une et **une seule** raison de changer.
Elle doit avoir une seule et une seule responsabilité (fonction)

?

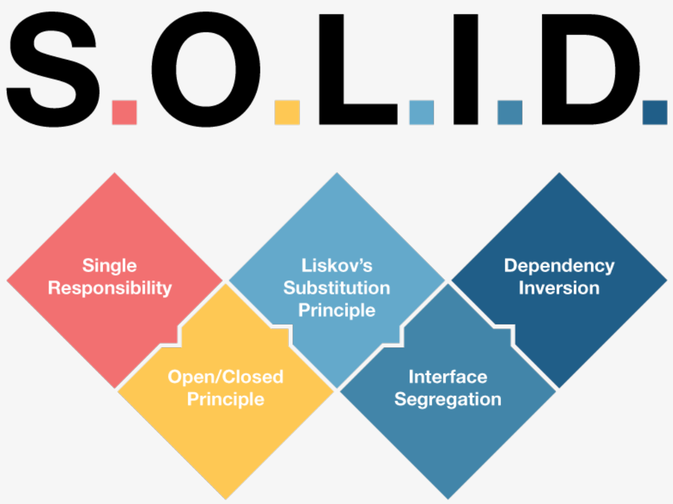
?

Class **AreaCalculator** -> self.shapes

02

PROGRAMMATION ORIENTÉE OBJET

SOLID FOR KNOWLEDGE



OPEN / CLOSED PRINCIPLE

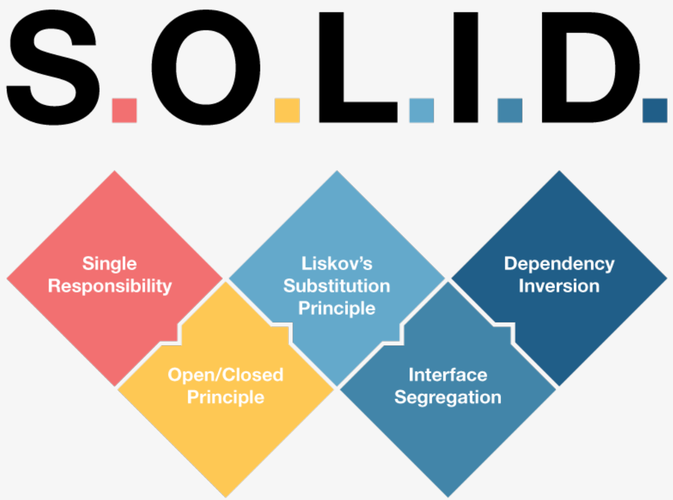
Les objets doivent être ouverts à l'extension, mais fermés à toute modification.

Ex : On ne remplace pas une chambre par un garage, on construit le garage

02

PROGRAMMATION ORIENTÉE OBJET

SOLID FOR KNOWLEDGE

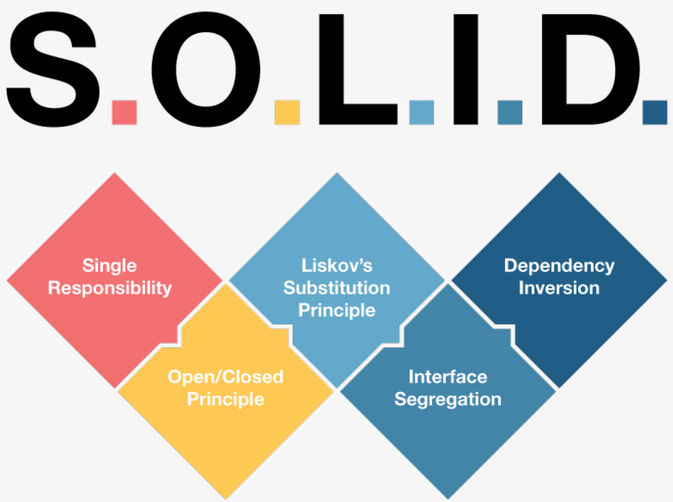


LISKOV'S SUBSTITUTION PRINCIPLE

Une classe peut-être remplacée par une sous-classe sans casser l'application

PROGRAMMATION ORIENTÉE OBJET

SOLID FOR KNOWLEDGE



INTERFACE SEGREGATION

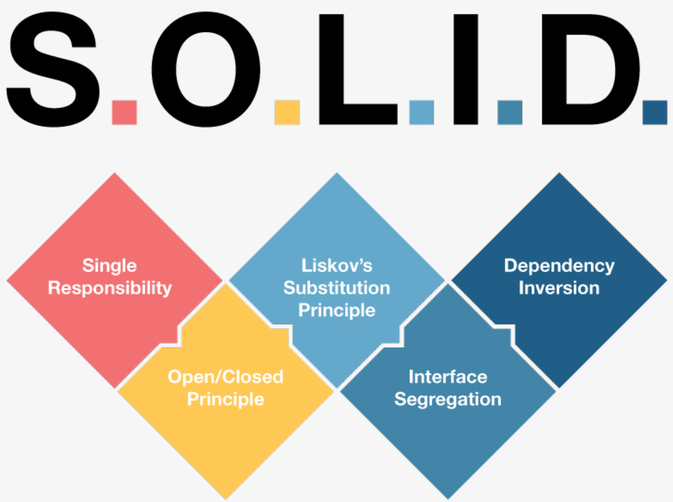
> Un client ne devrait pas être forcé d'implémenter une interface qu'il n'utilise pas.

Ex : La classe Square qui implémente CalculateVolume à cause de Shape

02

PROGRAMMATION ORIENTÉE OBJET

SOLID FOR KNOWLEDGE



DEPENDENCY INVERSION

Les objets doivent dépendre d'abstractions et pas de concrétions.

Ex : Password Reminder qui dépend de DB Connection

02

PROGRAMMATION ORIENTÉE OBJET

Garbage collector

- La **garbage collection** est automatique
- Python libère tout objet non-utilisé
- Un objet est supprimé lorsque son compteur de référence atteint 0

02 PROGRAMMATION ORIENTÉE OBJET

TP1

Créer une classe abstraite FileParser pour lire et écrire des fichiers. Créer ensuite les implémentations des classes filles pour les formats **txt**, **json**, **yaml** et **ini**

TP2

Créer un fichier **yaml** contenant des configurations de capteurs comme ci-dessous.

```yaml
sensors:
    Sensor_1:
        coordinates: [1, 1]
        unit: "%"
        type: "humidity"
        range: [0, 100]
    Sensor_2:
        coordinates: [1, 1]
        unit: "L"
        type: "light"
        range: [0, 100]
    Sensor_3:
        coordinates: [1, 1]
        unit: "C"
        type: "temperature"
        range: [0, 100]
    Sensor_4:
        coordinates: [1, 1]
        unit: "?"
        type: "motion"
        range: [0, 100]
```

Créer une fonction qui choisit un capteur aléatoire dans la liste et qui renvoie un **dictionnaire** comprenant les valeurs statiques du capteur et la valeur de lecture aléatoire dans **range**.

Vous utilisez les modules **pyaml** et **json** pour le parser.

INTRODUCTION À MQTT

03

03

INTRODUCTION À MQTT

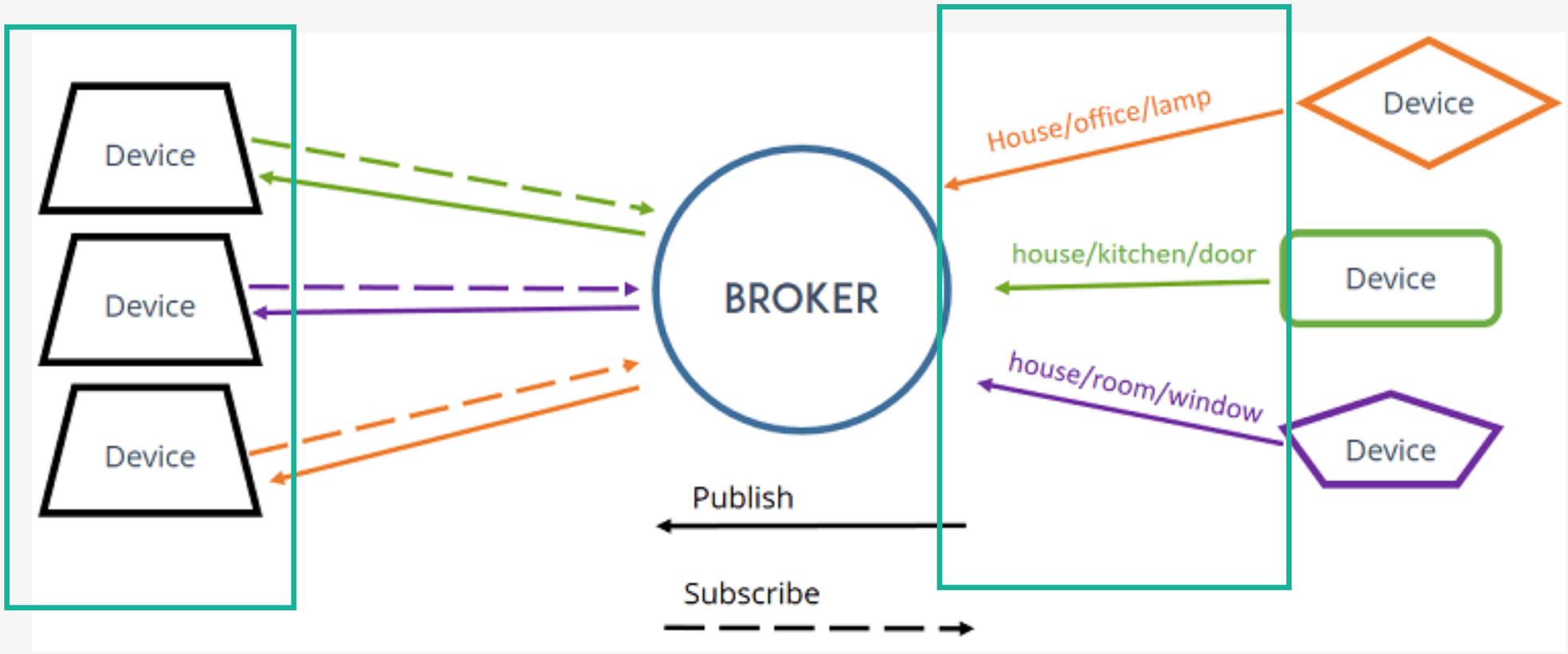
Terminologie

- Message Queuing Telemetry Transport
- **Open-source** et développé par IBM
- Décrit un **broker** (serveur / routeur / la poste)
- Modèle **Publish / Subscribe**
- Avantage d'être **lightweight** (donc adapté à l'IoT)

03

INTRODUCTION À MQTT

Terminologie



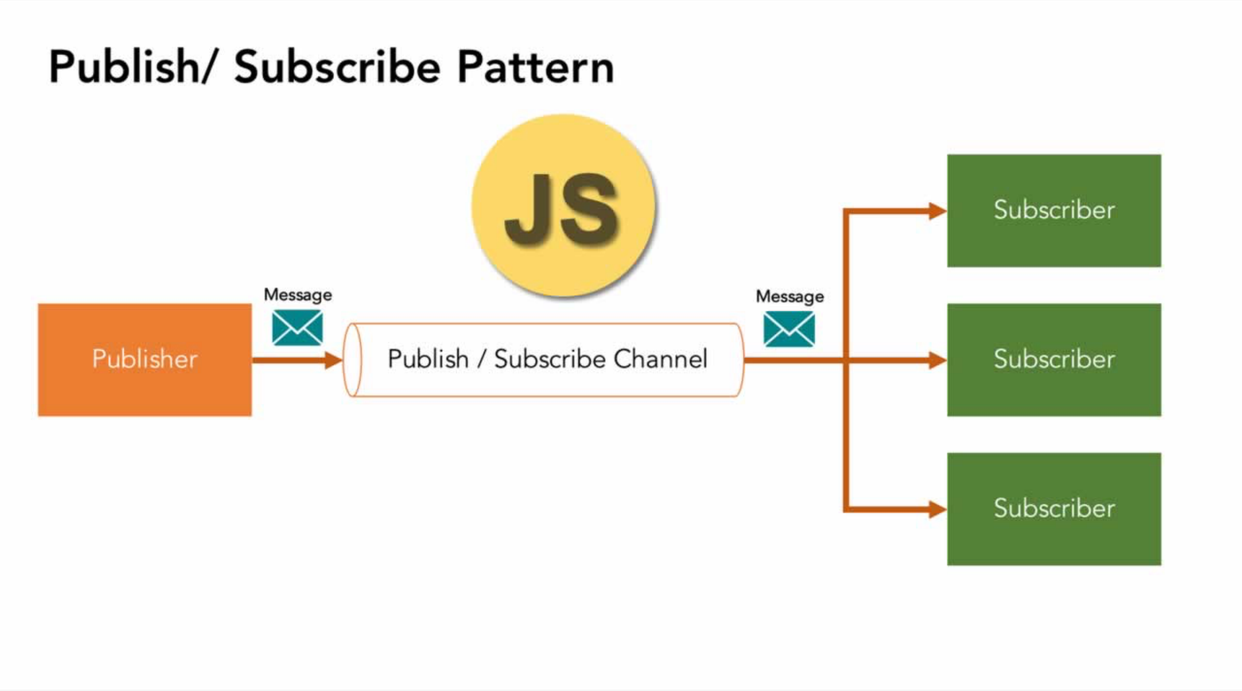
CLIENTS

TOPICS

03

INTRODUCTION À MQTT

Terminologie



Publish / Subscribe

- Base du protocol
- La relation client - topic est N to N
- Limitation de 260 MO par message (UTF-8)
- Validation de certificat et cryptage TLS
- Possibilité de fournir une ACL (Access list)

03 INTRODUCTION À MQTT

TP1 Créer une fonction qui se connecte au broker **broker.shiftr.io**

TP2 Créer une fonction qui **publish** sur n'importe quel topic du broker.

TP3 Créer une fonction qui subscribe au topic **try** du broker et affiche les messages en console

TP4 Coupler le TP précédent pour **publish** un payload correspondant à un capteur aléatoire présent dans votre fichier de capteur. Vous publierez une valeur de capteur toutes les secondes

AVATAR

04

04

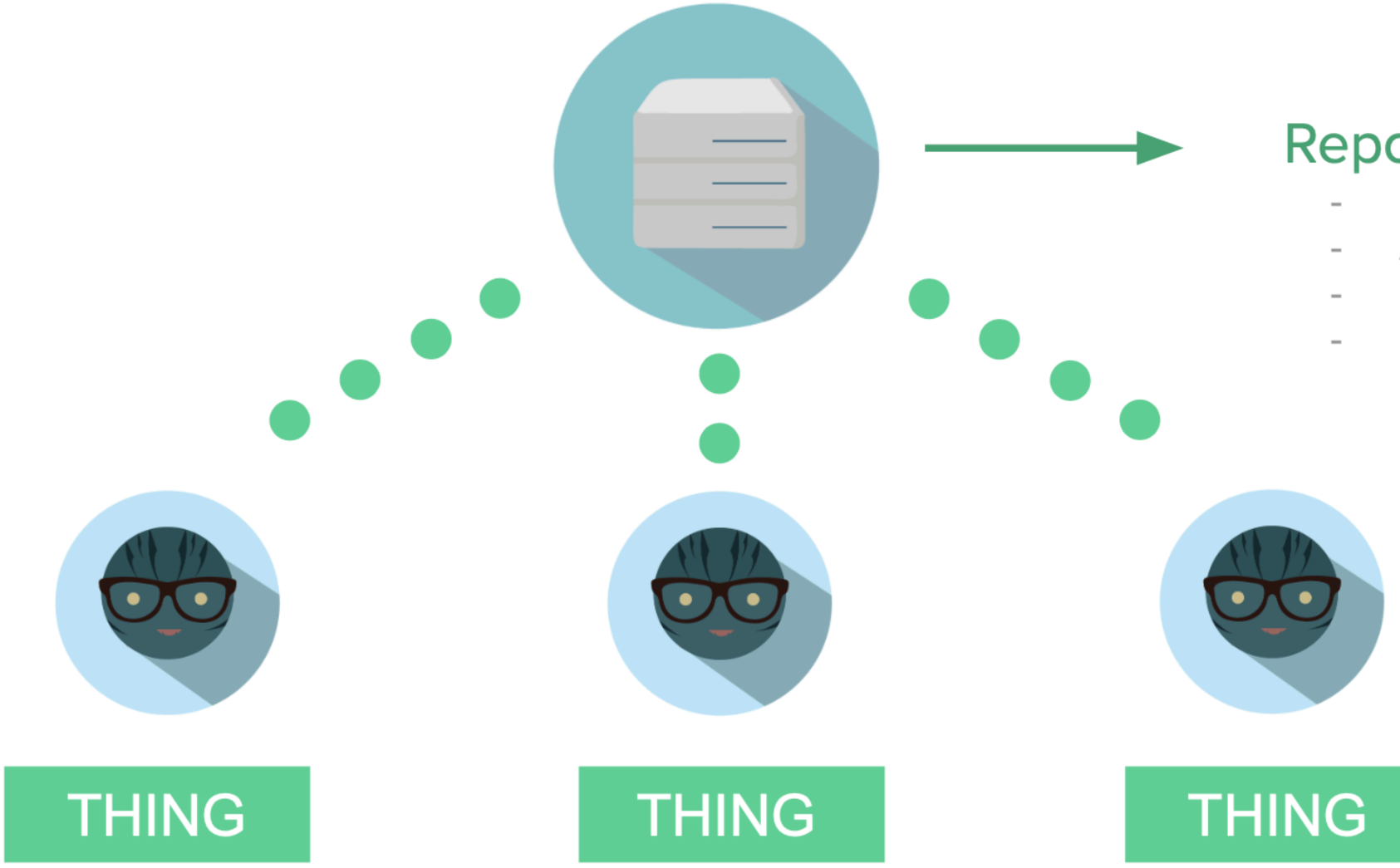
AVATAR

LE CONCEPT

- Représentation virtuelle d’un objet
- Connaissance ontologique
- Connaissance ontologique
- Réutilisabilité
- Agnostique au niveau de l’hébergement
- **Local :** SSDP, mDNS
- **Nearby :** BLE, RFID
- **Remote :** WIFI

Discovery

Communauté d'Avatars

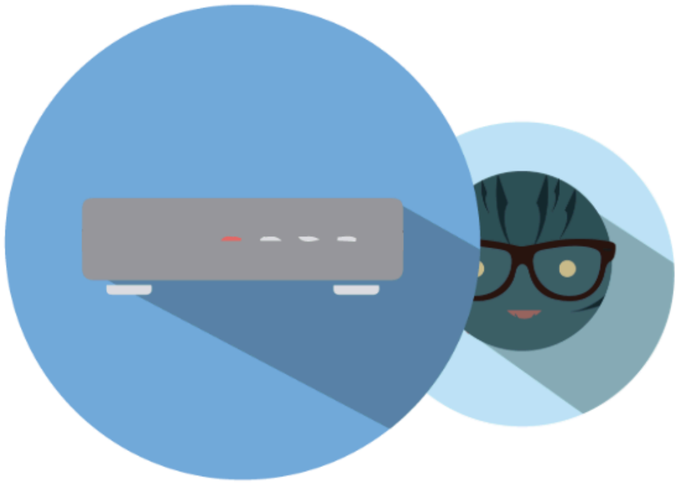


Repository

- "Annuaire" d'Avatars.
- Avatar identifiable par URI
- Query
- Renvoie les ontologies des objets

Disposition sur le réseau

THING



GATEWAY

CLOUD

1. Connection des objets sur réseau local

2. Création des avatars

3. Exposition des fonctionnalités (similarités)

4. Formation d'une communauté

5. L'avatar de Bob query le capteur de température & un service externe

6. Evaluation du meilleur scénario par le reasoner

7. Action

Use Case

Smart Home

Bob

Besoin : 21 °C

19 °C

24 °C

Repository

Ouverture

Fenêtre

Smart Home

Temp : 24 °C