Algorithme et Développement dans la Data

YNOV Toulouse - M1 Data & IA

Intervenant Enzo DARDAILLON

03/11/2025

Les structures de données élémentaires

Les structures de données élémentaires

	SQL (PostgreSQL)	Python
Booléen (Vrai / Faux)	boolean	bool
Nombre entier (0,1,2,3,4…)	integer	int
Nombre à virgule (3.14)	numeric	float
Caractère (A,B,c,d)	char(1)	str
Chaîne de caractères	varchar(n), text	str
Date	date, timestamp	date, datetime

Répartition de la mémoire

	SQL (PostgreSQL)	Python
Booléen (Vrai / Faux)	1 octet	28 octets
Nombre entier (0,1,2,3,4…)	4 octets	minimum 28 octets
Nombre à virgule (3.14)	8 octets	24 octets
Caractère (A,B,c,d)	1 octet	minimum 41 octets
Chaîne de caractères	1 + n + 1 octets	minimum 41 octets
Date	4 ou 8 octets	48 octets

Petit bonus :

Avez-vous déjà casté un String en Integer sous Python ?

Les ensembles de structures de données : L'approche Objet

Classe Station

- Nom → Chaîne de caractères
- Humidité → Nombre entier

Application “Météo”

Python

```python
class Station:
  def __init__(self, nom: str, humidite: int) -> None:
    self.nom: str = nom
    self.humidite: int = humidite

#Les annotations sont uniquement pour la documentation
```

PostgreSQL

```sql
CREATE TABLE STATION (
  id INT PRIMARY KEY AUTO_INCREMENT,
  nom VARCHAR(64),
  humidite INT
);
```

Classe Ville

- Nom → Chaîne de caractères
- Stations → Liste de stations

Application “Météo”

Python

```python
class Ville:
  def __init__(self, nom: str) -> None:
    self.nom: str = nom
    self.stations: list[Stations] = []

# Les annotations sont uniquement pour la documentation
```

PostgreSQL

```sql
CREATE TABLE VILLE (
  id INT PRIMARY KEY AUTO_INCREMENT,
  nom VARCHAR(64)
);

-- Le lien SQL entre STATION et VILLE n’est pas étudié
dans ce module
```

A vous de jouer !

Créez une méthode dans Ville pour ajouter ou supprimer une station

Python

```python
class Ville:
  def __init__(self, nom: str) -> None:
    self.nom: str = nom
    self.stations: list[Stations] = []

# Les annotations sont uniquement pour la documentation
```

Héritage

- Classe parent
- Classe enfant
- Transmission des attributs / méthodes

Application “Zoo”

Python

```python
class Animal:
  def __init__(self, nom: str) -> None:
    self.nom: str = nom

  def manger(self) -> None:
    print("Je mange")
```

Python

```python
class Chien(Animal):
  def __init__(self, nom: str, race: str ) -> None:
    super().__init__(nom)
    self.race: str = race

#  def manger(self) -> None:
#    print("Je mange")

  def sentir(self) -> None:
    print(f"{self.nom} sent une fleur")
```

Python

```python
mon_chien = Chien("Rex", "Labrador")

mon_chien.manger() # Valide pour tout objet Animal
mon_chien.sentir() # Valide uniquement pour Chien
```

Clean Code

Principe SOLID

Clean Code

- **S**ingle responsibility
  - Une classe ou une méthode ne doit avoir qu'une seule raison d'exister
- **O**pen / Closed
  - Une classe ne doit jamais être modifiée, elle peut juste être étendue
- **L**iskov substitution
  - Une méthode ne doit pas être entraînée d'erreur à cause de l'héritage
- **I**nterface Segregation
  - On préfère plusieurs interfaces à usage unique plutôt qu'une grosse interface multitâche
- **D**ependency inversion
  - Une classe fille dépend de sa mère
  - Une classe mère ne dépend pas de sa fille

SOLID - Single responsibility

```python
def process_data(df):
  df['Total'] = df['Quantité'] * df['Prix']
  print(df)

  return df
```

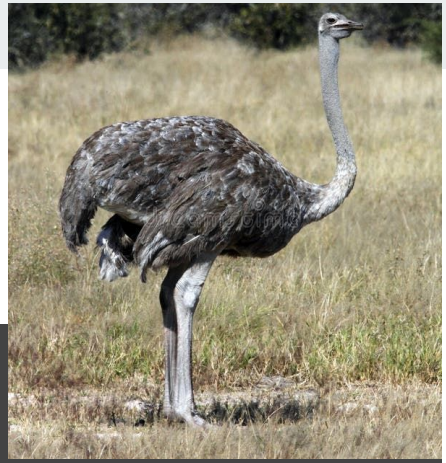
```python
def process_data(df):
  df['Total'] = df['Quantité'] * df['Prix']

  return df


def show_data(df):
  print(df)
```

SOLID - Open / Closed

```python
class DataAnalyzer:
  def __init__(self, data):
    self.df = pd.DataFrame(data)

  def calculate_average(self):
    return self.df['Valeurs'].mean()

class DataAnalyzerWithMedian(DataAnalyzer):
  def calculate_median(self):
    return self.df['Valeurs'].median()

data = {'Valeurs': [10, 20, 30, 40]}

analyzer = DataAnalyzer(data)
average = analyzer.calculate_average()
print(f"Moyenne: {average}")

analyzer_with_median = DataAnalyzerWithMedian(data)
median = analyzer_with_median.calculate_median()
print(f"Médiane: {median}")
```

```python
class DataAnalyzer:
  def __init__(self, data):
    self.df = pd.DataFrame(data)

  def calculate(self):
    pass

class AverageAnalyzer(DataAnalyzer):
  def calculate(self):
    return self.df['Valeurs'].mean()

class MedianAnalyzer(DataAnalyzer):
  def calculate(self):
    return self.df['Valeurs'].median()

data = {'Valeurs': [10, 20, 30, 40]}

average_analyzer = AverageAnalyzer(data)
average = average_analyzer.calculate()
print(f"Moyenne: {average}")

median_analyzer = MedianAnalyzer(data)
median = median_analyzer.calculate()
print(f"Médiane: {median}")
```

SOLID - **Liskov substitution**



```python
class Oiseau:
  def voler(self):
    return "Je vole!"

class Moineau(Oiseau):
  pass

class Autruche(Oiseau):
  def voler(self):
    raise Exception("L'autruche ne peut pas voler.")


Moineau().voler() # Ça fonctionne
Autruche().voler() # Ça va causer une erreur
```

```python
class Oiseau:
  pass

class OiseauVolant(Oiseau):
  def voler(self):
    return "Je vole!"

class Moineau(OiseauVolant):
  pass

class Autruche(Oiseau):
  pass


Moineau().voler() # Ça fonctionne
Autruche().voler() # Cette méthode n'existe pas
```

SOLID - **Interface Segregation**

```python
class IMachine:
  def print(self):
    pass

  def fly(self):
    pass

class Drone(IMachine):
  def print(self):
    print("Printing from the drone")

  def fly(self):
    print("Flying drone")

class Printer(IMachine):
  def print(self):
    print("Printing from the printer")

  def fly(self):
    raise NotImplementedError("Printers cannot fly")
```

```python
class IPrintable:
  def print(self):
    pass

class IFlyable:
  def fly(self):
    pass

class Drone(IFlyable, IPrintable):
  def print(self):
    print("Printing from the drone")

  def fly(self):
    print("Flying drone")

class Printer(IPrintable):
  def print(self):
    print("Printing from the printer")
```

SOLID - Dependency inversion

```python
class EmailService:
  def send_email(self, recipient, message):
    print(f"Email envoyé à {recipient} : {message}")

class SMSService:
  def send_sms(self, recipient, message):
    print(f"SMS envoyé à {recipient} : {message}")

class NotificationSystem:
  def __init__(self):
    self.email_service = EmailService()
    self.sms_service = SMSService()

  def notify_via_email(self, recipient, message):
    self.email_service.send_email(recipient, message)

  def notify_via_sms(self, recipient, message):
    self.sms_service.send_sms(recipient, message)

NotificationSystem().notify_via_email("a@example.com", "Hey!")
NotificationSystem().notify_via_sms("0612345678", "Salut!")
```

```python
class NotificationService:
  def send(self, recipient, message):
    pass

class EmailService(NotificationService):
  def send(self, recipient, message):
    print(f"Email envoyé à {recipient} : {message}")

class SMSService(NotificationService):
  def send(self, recipient, message):
    print(f"SMS envoyé à {recipient} : {message}")

class NotificationSystem:
  def __init__(self, service: NotificationService):
    self.service = service

  def notify(self, recipient, message):
    self.service.send(recipient, message)

NotificationSystem(EmailService()).notify("a@example.com", "Hey!")
NotificationSystem(SMSService()) .notify("0612345678", "Salut!")
```

Principe KISS

Keep **I**t **S**imple **S**tupid
Gardez votre code simple et stupide

Clean Code

```python
def multiplication(a, b):
  # Gérer les cas où l'un des opérandes est zéro
  if a == 0 or b == 0:
    return 0

  # Utiliser l'addition répétée
  result = 0
  for _ in range(abs(b)):
    result += abs(a)

  # Gérer le signe du résultat
  if (a < 0) ^ (b < 0):
    result = -result

  return result
```

```python
def multiplication2(a, b):
  return a*b
```

Principe DRY

Don't **R**epeat **Y**ourself
Ne vous répétez pas !

(Ne réinventez pas la roue)

Clean Code

```python
def multiplication(a, b):
  # Gérer les cas où l'un des opérandes est zéro
  if a == 0 or b == 0:
    return 0

  # Utiliser l'addition répétée
  result = 0
  for _ in range(abs(b)):
    result += abs(a)

  # Gérer le signe du résultat
  if (a < 0) ^ (b < 0):
    result = -result

  return result
```

```python
def multiplication2(a, b):
  return a*b
```

Principe YAGNI

You **A**ren't **G**onna **N**eed **I**t
Vous n'allez pas en avoir besoin

N'allez pas au-delà du cahier des charges

Clean Code

```python
def multiplication(a, b):
  # Gérer les cas où l'un des opérandes est zéro
  if a == 0 or b == 0:
    return 0

  # Utiliser l'addition répétée
  result = 0
  for _ in range(abs(b)):
    result += abs(a)

  # Gérer le signe du résultat
  if (a < 0) ^ (b < 0):
    result = -result

  return result
```

```python
def multiplication2(a, b):
  return a*b
```

Exigences non-fonctionnelles

Exigences non-fonctionnelles

- Documentation → (YDATA Profiling)
- Intégrité des données → Tests de la qualité des données (Pydantic)
- Sécurité → Tests d'intrusion…
- Protection de l'environnement → Audit à la qualité de l'environnement
- Sauvegarde → Versionning (Git)

- Voir +

En route vers le TP !
(Application Météo)

Sources et outils pratiques

https://en.wikipedia.org/wiki/Non-functional_requirement

https://docs.profiling.ydata.ai/latest/

https://docs.pydantic.dev/latest/

https://fr.wikipedia.org/wiki/SOLID_%28informatique%29

https://moodle.learn.ynov.com/course/view.php?id=73970

https://data.toulouse-metropole.fr/explore/?sort=modified&refine.modified=2025&refine.keyword=m%C3%A9t%C3%A9o