

Langage et algorithme

CR6. Type Abstrait de Données (TAD)

A. Malek TOUMI

toumiab@ensta-bretagne.fr

2017/2018

ENSTA Bretagne



Sommaire

1 Type abstrait de données

Définition

2 Notion d'objet

Concepts fondamentaux

POO en Python

3 Retour sur la notion de TAD

TAD Intervalle

TAD Pile

TAD File

Autres TAD

4 Horloge



Sommaire

1 Type abstrait de données

Définition

2 Notion d'objet

Concepts fondamentaux

POO en Python

3 Retour sur la notion de TAD

TAD Intervalle

TAD Pile

TAD File

Autres TAD

4 Horloge



Définition

[toggle](#)[reset](#)

Définition (Type abstrait de données)

Concept d'utilisation de données ne tenant pas en compte leur représentation interne.



Définition

[toggle](#)[reset](#)

Définition (Type abstrait de données)

Concept d'utilisation de données ne tenant pas en compte leur représentation interne.

- Conception d'un algorithme : démarche descendante



Définition

[toggle](#)[reset](#)

Définition (Type abstrait de données)

Concept d'utilisation de données ne tenant pas en compte leur représentation interne.

- Conception d'un algorithme : démarche descendante
- Initialement : on est loin de l'implantation...



Définition

[toggle](#)[reset](#)

Définition (Type abstrait de données)

Concept d'utilisation de données ne tenant pas en compte leur représentation interne.

- Conception d'un algorithme : démarche descendante
- Initialement : on est loin de l'implantation... puis on affine



Définition

[toggle](#)[reset](#)

Définition (Type abstrait de données)

Concept d'utilisation de données ne tenant pas en compte leur représentation interne.

- Conception d'un algorithme : démarche descendante
- Initialement : on est loin de l'implantation... puis on affine
- Étape initiale : type abstrait de données (TAD)



Définition

[toggle](#)[reset](#)

Définition (Type abstrait de données)

Concept d'utilisation de données ne tenant pas en compte leur représentation interne.

- Conception d'un algorithme : démarche descendante
- Initialement : on est loin de l'implantation... puis on affine
- Étape initiale : type abstrait de données (TAD)
- Définir les opérations sur le TAD
- Cacher la représentation interne



Définition

[toggle](#)[reset](#)

Définition (Type abstrait de données)

Concept d'utilisation de données ne tenant pas en compte leur représentation interne.

- Conception d'un algorithme : démarche descendante
- Initialement : on est loin de l'implantation... puis on affine
- Étape initiale : type abstrait de données (TAD)
- Définir les opérations sur le TAD
- Cacher la représentation interne
- Objet simplifié



Définition

[toggle](#)[reset](#)

Définition (Type abstrait de données)

Concept d'utilisation de données ne tenant pas en compte leur représentation interne.

- Conception d'un algorithme : démarche descendante
- Initialement : on est loin de l'implantation... puis on affine
- Étape initiale : type abstrait de données (TAD)
- Définir les opérations sur le TAD
- Cacher la représentation interne
- Objet simplifié

Utilité : briques réutilisables



Type abstrait de données

Exemple : rationnels

toggle

reset

- TAD nombre rationnel



Exemple : rationnels

toggle

reset

- TAD nombre rationnel
- Opérations :



Exemple : rationnels

toggle

reset

- TAD nombre rationnel
- Opérations :
 - création



Exemple : rationnels

toggle

reset

- TAD nombre rationnel
- Opérations :
 - création
 - extraction numérateur et dénominateur



Exemple : rationnels

toggle

reset

- TAD nombre rationnel
- Opérations :
 - création
 - extraction numérateur et dénominateur
 - addition, soustraction, multiplication, division



Exemple : rationnels

toggle

reset

- TAD nombre rationnel
- Opérations :
 - création
 - extraction numérateur et dénominateur
 - addition, soustraction, multiplication, division
 - test d'égalité



Exemple : rationnels

toggle

reset

- TAD nombre rationnel
- Opérations :
 - création
 - extraction numérateur et dénominateur
 - addition, soustraction, multiplication, division
 - test d'égalité
- Représentation interne : par exemple deux entiers



Exemple : rationnels

[toggle](#)[reset](#)

- TAD nombre rationnel
- Opérations :
 - création
 - extraction numérateur et dénominateur
 - addition, soustraction, multiplication, division
 - test d'égalité
- Représentation interne : par exemple deux entiers
- Contrainte interne : dénominateur strictement positif



Exemple : rationnels

toggle

reset

- Définition d'un nouveau **type**
- Classe

```
class Rationnel (object):
```



Exemple : rationnels

toggle

reset

- Représentation **interne**
- Constructeur

```
class Rationnel (object):  
    def __init__(self, num, den = 1):
```



Exemple : rationnels

[toggle](#)[reset](#)

- Initialisation de la représentation interne
- Constructeur et Variables d'instance

```
class Rationnel (object):  
    def __init__(self, num, den = 1):  
        self.__num = num  
        self.__den = den
```



Exemple : rationnels

[toggle](#)[reset](#)

- Initialisation de la représentation interne
- Constructeur et Variables d'instance

```
class Rationnel (object):  
    def __init__(self, num, den = 1):  
        if (den == 0):  
            self.__num = 0  
            self.__den = 0  
        elif den < 0:  
            self.__num = - num  
            self.__den = - den
```



Exemple : rationnels

[toggle](#)[reset](#)

- Initialisation de la représentation interne
- Constructeur et Variables d'instance

```
class Rationnel (object):
    def __init__(self, num, den = 1):
        if (den == 0):
            self.__num = 0
            self.__den = 0
        elif den < 0:
            self.__num = - num
            self.__den = - den
        else :
            self.__num = num
            self.__den = den
```



Exemple : rationnels

[toggle](#)[reset](#)

- Opérations
- Méthodes d'instance : **égalité**, multiplication, représentation

```
class Rationnel (object):  
  
    def __eq__(self, other):  
        return self.__num * other.__den == self.__den  
        * other.__num
```



Exemple : rationnels

[toggle](#)[reset](#)

- Opérations
- Méthodes d'instance : égalité, multiplication, représentation

```
class Rationnel (object):  
  
    def __eq__(self, other):  
        return self.__num * other.__den == self.__den  
            * other.__num  
    def __mult__(self, other):  
        n = self.__num * other.__num  
        d = self.__den * other.__den  
        return Rationnel(n, d)
```



Exemple : rationnels

[toggle](#)[reset](#)

- Opérations
- Méthodes d'instance : égalité, multiplication, **représentation**

```
class Rationnel (object):  
  
    def __mult__(self, other):  
        n = self.__num * other.__num  
        d = self.__den * other.__den  
        return Rationnel(n, d)  
    def __str__(self):  
        return '{0}/{1}'.format(self.__num, self.__den)
```



Exemple : rationnels

toggle

reset

Exemple

```
# Test : création de rationnels
r1 = Rationnel(1, -2)
r2 = Rationnel(3, 5)
# Affichage des rationnels
print(r1, r2, r1*r2, r1+r2)
```



Exemple : rationnels

[toggle](#)[reset](#)

Exemple

```
# Test : création de rationnels
r1 = Rationnel(1, -2)
r2 = Rationnel(3, 5)
# Affichage des rationnels
print(r1, r2, r1*r2, r1+r2)
# => -1/2 3/5 -3/10 1/10
```



Exemple : rationnels

[toggle](#)[reset](#)

Exemple

```
# Test : création de rationnels
r1 = Rationnel(1, -2)
r2 = Rationnel(3, 5)
# Affichage des rationnels
print(r1, r2, r1*r2, r1+r2)
# => -1/2 3/5 -3/10 1/10
# Test d'égalité de rationnels
print(r1 == Rationnel(-2, 4))
```



Exemple : rationnels

[toggle](#)[reset](#)

Exemple

```
# Test : création de rationnels
r1 = Rationnel(1, -2)
r2 = Rationnel(3, 5)
# Affichage des rationnels
print(r1, r2, r1*r2, r1+r2)
# => -1/2 3/5 -3/10 1/10
# Test d'égalité de rationnels
print(r1 == Rationnel(-2, 4)) # => True
```



Exemple : rationnels

[toggle](#)[reset](#)

```
Rationnel
+-- num
+-- den
+-- init_
+-- eq_
+-- add_
+-- mul_
+-- str_
+-- get_numerateur
+-- get_denominateur
+-- pgcd
+-- simplifie
```



Définition

Remarques (Quelques remarques importantes)

- Tous les attributs et méthodes des classes Python sont publics au sens de java et C++,
- Le constructeur d'une classe est une méthode spéciale qui s'appelle `__init__(self)`.
- Les attributs (variables d'instance) sont créés et initialisés dans le constructeur (`self.__nomVar, self._nomVar, self.nomVar`)
- Toutes les méthodes d'instance prennent une variable `self` comme premier argument. Cette variable est une référence à l'objet manipulé.



Sommaire

1 Type abstrait de données

Définition

2 Notion d'objet

Concepts fondamentaux

POO en Python

3 Retour sur la notion de TAD

TAD Intervalle

TAD Pile

TAD File

Autres TAD

4 Horloge



Sommaire

1 Type abstrait de données

Définition

2 Notion d'objet

Concepts fondamentaux

POO en Python

3 Retour sur la notion de TAD

TAD Intervalle

TAD Pile

TAD File

Autres TAD

4 Horloge



Concepts

[toggle](#)[reset](#)

Objectif :

- Augmenter la réutilisabilité
- ⇒ modularité du code, notion de composants



Objectif :

- Augmenter la réutilisabilité
- ⇒ modularité du code, notion de composants

Concepts mis en place :

- encapsulation
- héritage
- polymorphisme



Objectif :

- Augmenter la réutilisabilité
- ⇒ modularité du code, notion de composants

Concepts mis en place :

- encapsulation
- héritage
- polymorphisme



Concepts

[toggle](#)[reset](#)

Objectif :

- Augmenter la réutilisabilité
- ⇒ modularité du code, notion de composants

Concepts mis en place :

- encapsulation
- héritage
- polymorphisme

Définition (Encapsulation)

Mécanisme permettent de cacher l'implantation d'un TAD. Fournit une interface pour accéder aux données.



Notion de classe et d'instance

- Classe \simeq type
- Instancier la classe \simeq créer une variable de ce type
- Objet ou instance : variable créée dont le type est une classe



Notion de classe et d'instance

- Classe \simeq type
- Instancier la classe \simeq créer une variable de ce type
- Objet ou instance : variable créée dont le type est une classe

Classe \leadsto plan

Instance \leadsto réalisation



Notion de classe et d'instance

- Classe \simeq type
- Instancier la classe \simeq créer une variable de ce type
- Objet ou instance : variable créée dont le type est une classe

Classe \leadsto plan

Instance \leadsto réalisation

Important

Un programme est composé d'instances (objets) et non de classes



Exemple

- Classe Rationnel



Exemple

Classe Rationnel

```
x = Rationnel(2,3)
```



Exemple

- Classe Rationnel
- Création d'instances / objets



Exemple

Classe Rationnel

```
__num=2  
__den=3  
__mult__()  
__eq__()
```



```
x = Rationnel(2,3)
```



Exemple

Classe Rationnel

`__num=2
__den=3
__mult__()
__eq__()`

`__num=1
__den=5
__mult__()
__eq__()`



```
x = Rationnel(2,3)  
y = Rationnel(1,5)
```



Exemple

- Classe Rationnel
- Création d'instances / objets
- Utilisation d'objets : exemple $x = x*y$

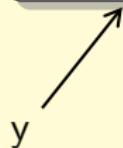


Exemple

Classe Rationnel

`__num=2`
`__den=3`
`__mult__()`
`__eq__()`

`__num=1`
`__den=5`
`__mult__()`
`__eq__()`



```
x = Rationnel(2,3)  
y = Rationnel(1,5)
```



Sommaire

1 Type abstrait de données

Définition

2 Notion d'objet

Concepts fondamentaux

POO en Python

3 Retour sur la notion de TAD

TAD Intervalle

TAD Pile

TAD File

Autres TAD

4 Horloge



Notion de classe et d'objet en Python

toggle

reset

- plusieurs classes peuvent être définies dans un seul fichier (module) **.py**
- Nom de la classe dans le fichier après le **class**
- Usage : faire commencer le nom de la classe par une majuscule



Notion de classe et d'objet en Python

toggle

reset

- plusieurs classes peuvent être définies dans un seul fichier (module) **.py**
- Nom de la classe dans le fichier après le **class**
- Usage : faire commencer le nom de la classe par une majuscule



Instanciation de classes

toggle

reset

- Utiliser une classe \Rightarrow créer une instance
- Instance = variable \Rightarrow se manipule comme une variable

Exemple



Instanciation de classes

[toggle](#)[reset](#)

- Utiliser une classe \Rightarrow créer une instance
- Instance = variable \Rightarrow se manipule comme une variable
 - création et initialisation

Exemple

```
x = Rationnel(2,3)
```



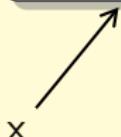
Instanciation de classes

toggle

reset

Classe Rationnel

```
__num=2  
__den=3  
__mult__()  
__eq__()
```





Instanciation de classes

[toggle](#)[reset](#)

- Utiliser une classe \Rightarrow créer une instance
- Instance = variable \Rightarrow se manipule comme une variable
 - création et initialisation
 - utilisation

Exemple

```
x = Rationnel(2,3)
x = x * x # x = x.__mul__(x)
```



Instanciation de classes

[toggle](#)[reset](#)

Classe Rationnel

```
__num=2  
__den=3  
__mult__()  
__eq__()
```





Variables d'instance

[toggle](#)[reset](#)

- Attributs de la classe : créées et initialisées dans le constructeur et attachées à la variable `self`

Exemple (Rationnels)

```
class Rationnel (object):
    def __init__(self, num, den = 1):
        # deux variables d'instance
        self.__num = num
        self.__den = den
```



Variables d'instance

[toggle](#)[reset](#)

- Attributs de la classe : créées et initialisées dans le constructeur et attachées à la variable `self`
- En général, pas accessibles directement et renommées en interne
⇒ `__attribut`

Exemple (Rationnels)

```
class Rationnel (object):  
    def __init__(self, num, den = 1):  
        # deux variables d'instance  
        self.__num = num  
        self.__den = den
```



Variables d'instance

[toggle](#)[reset](#)

- Attributs de la classe : créées et initialisées dans le constructeur et attachées à la variable `self`
- En général, pas accessibles directement et renommées en interne
⇒ `__attribut`
- Règle de nommage : nom commence par une minuscule

Exemple (Rationnels)

```
class Rationnel (object):
    def __init__(self, num, den = 1):
        # deux variables d'instance
        self.__num = num
        self.__den = den
```



Visibilité des attributs

[toggle](#)[reset](#)

Remarque

Tous les attributs et méthodes des classes Python sont publics au sens de java et C++,

Par convention :

- Une variable dont le nom commence par (_) \Rightarrow est considérée comme **private**
- Une variable dont le nom commence par (--) \Rightarrow est considérée **private** et renommée en interne
- Autrement \Rightarrow variable **public**



Visibilité des attributs

[toggle](#)[reset](#)

Exemple

```
class Personne(object):
    def __init__(self):
        self.nom = "inconnu" # Variable publique
        self._prenom = "xxx" # Variable privée
        self.__adr = "Brest" # Variable privée,
                            # =>: __Personne__adr #
```

Exécution

```
p = Personne()
print(p.nom)
```



Visibilité des attributs

[toggle](#)[reset](#)

Exemple

```
class Personne(object):
    def __init__(self):
        self.nom = "inconnu" # Variable publique
        self._prenom = "xxx" # Variable privée
        self.__adr = "Brest" # Variable privée,
                            # =>: __Personne__adr #
```

Exécution

```
p = Personne()
print(p.nom) # -> 'inconnu'
print(p._prenom)
```



Visibilité des attributs

[toggle](#)[reset](#)

Exemple

```
class Personne(object):
    def __init__(self):
        self.nom = "inconnu" # Variable publique
        self._prenom = "xxx" # Variable privée
        self.__adr = "Brest" # Variable privée,
                            # =>: _Personne__adr #
```

Exécution

```
p = Personne()
print(p.nom) # -> 'inconnu'
print(p._prenom) # ->'xxx' : non recommandé
```



Visibilité des attributs

[toggle](#)[reset](#)

Exemple

```
class Personne(object):
    def __init__(self):
        self.nom = "inconnu" # Variable publique
        self._prenom = "xxx" # Variable privée
        self.__adr = "Brest" # Variable privée,
                            # =>: _Personne__adr #
```

Exécution

```
p = Personne()
print(p.nom) # -> 'inconnu'
print(p._prenom) # ->'xxx' : non recommandé
print(p.__adr)
```



Visibilité des attributs

[toggle](#)[reset](#)

Exemple

```
class Personne(object):
    def __init__(self):
        self.nom = "inconnu" # Variable publique
        self._prenom = "xxx" # Variable privée
        self.__adr = "Brest" # Variable privée,
                            # =>: _Personne__adr #
```

Exécution

```
p = Personne()
print(p.nom) # -> 'inconnu'
print(p._prenom) # ->'xxx' : non recommandé
print(p.__adr) # -> Erreur
```



Visibilité des attributs

[toggle](#)[reset](#)

Exemple

```
class Personne(object):
    def __init__(self):
        self.nom = "inconnu" # Variable publique
        self._prenom = "xxx" # Variable privée
        self.__adr = "Brest" # Variable privée,
                            # =>: _Personne__adr #
```

Exécution

```
p = Personne()
print(p.nom) # -> 'inconnu'
print(p._prenom) # ->'xxx' : non recommandé
print(p.__adr) # -> Erreur
print(p._Personne__adr) # -> 'Brest'
```



Accesseurs

toggle

reset

Encapsulation

- Protéger l'accès aux variables d'instance : comment y accéder ?



Accesseurs

toggle

reset

Encapsulation

- Protéger l'accès aux variables d'instance : comment y accéder ?
 - ➊ Utilisation de méthodes get et set



Accesseurs

[toggle](#)[reset](#)

Encapsulation

- Protéger l'accès aux variables d'instance : comment y accéder ?
 - ➊ Utilisation de méthodes get et set
 - ➋ Utilisation de décorateurs

Exemple (get() et set())

```
class Rationnel(object) :  
    def __init__(self, num, den =1) :  
        self.__num = num
```



Accesseurs

[toggle](#)[reset](#)

Encapsulation

- Protéger l'accès aux variables d'instance : comment y accéder ?
 - ➊ Utilisation de méthodes get et set
 - ➋ Utilisation de décorateurs

Exemple (get() et set())

```
class Rationnel(object) :  
    def __init__(self, num, den =1) :  
        self.__num = num  
    def get_numerateur(self) :  
        return self.__num
```



Accesseurs

[toggle](#)[reset](#)

Encapsulation

- Protéger l'accès aux variables d'instance : comment y accéder ?
 - ➊ Utilisation de méthodes `get` et `set`
 - ➋ Utilisation de décorateurs

Exemple (`get()` et `set()`)

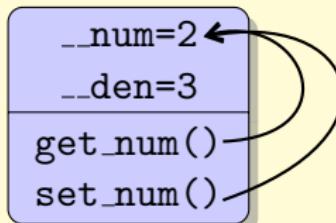
```
class Rationnel(object) :  
    def __init__(self, num, den =1) :  
        self.__num = num  
    def get_numerateur(self) :  
        return self.__num  
    def set_numerateur(self, num) :  
        self.__num = num
```



Accesseurs

[toggle](#)[reset](#)

Classe Rationnel



```
x = Rationnel(2,3)  
x.set_num(10)
```



Encapsulation

- Protéger l'accès aux variables d'instance : comment y accéder ?
 - ① utilisation de méthodes get et set
 - ② utilisation de décorateurs : remplaceget() et replaceSetter() :
 - remplace le getteur d'un attribut par une méthode devant laquelle figure `@property`
 - remplace le setteur d'un attribut par une méthode devant laquelle figure `@nomAttribut.setter`



Décorateur

[toggle](#)[reset](#)

Exemple

```
class Rationnel(object):
    def __init__(self, num, den =1):
        self.__num = num
    @property
    def num(self):
        return self.__num
    @num.setter
    def num(self, num):
        self.__num = num
```



Exemple

```
class Rationnel(object):
    def __init__(self, num, den =1):
        self.__num = num
    @property
    def num(self):
        return self.__num
    @num.setter
    def num(self, num):
        self.__num = num
# Exécution -----
r = Rationnel(2,5) # création de l'objet r =2/5
```



Décorateur

[toggle](#)[reset](#)

Exemple

```
class Rationnel(object):
    def __init__(self, num, den =1):
        self.__num = num
    @property
    def num(self):
        return self.__num
    @num.setter
    def num(self, num):
        self.__num = num
# Exécution -----
r = Rationnel(2,5) # création de l'objet r =2/5
print(r.num)
```



Décorateur

[toggle](#)[reset](#)

Exemple

```
class Rationnel(object):
    def __init__(self, num, den =1):
        self.__num = num
    @property
    def num(self):
        return self.__num
    @num.setter
    def num(self, num):
        self.__num = num
# Exécution -----
r = Rationnel(2,5) # création de l'objet r =2/5
print(r.num) # => acces à num de l'objet r : affiche 2
r.num = 10
```



Décorateur

[toggle](#)[reset](#)

Exemple

```
class Rationnel(object):
    def __init__(self, num, den =1):
        self.__num = num
    @property
    def num(self):
        return self.__num
    @num.setter
    def num(self, num):
        self.__num = num
# Exécution -----
r = Rationnel(2,5) # création de l'objet r =2/5
print(r.num) # => acces à num de l'objet r : affiche 2
r.num = 10 # => appel au setter : num vaut 10
print(r.num)
```



Décorateur

[toggle](#)[reset](#)

Exemple

```
class Rationnel(object):
    def __init__(self, num, den =1):
        self.__num = num
    @property
    def num(self):
        return self.__num
    @num.setter
    def num(self, num):
        self.__num = num
# Exécution -----
r = Rationnel(2,5) # création de l'objet r =2/5
print(r.num) # => acces à num de l'objet r : affiche 2
r.num = 10 # => appel au setter : num vaut 10
print(r.num) # => acces à num de l'objet r : affiche 10
```



Constructeur

[toggle](#)[reset](#)

- Rôle : créer et initialiser l'objet ([variables d'instance](#))
- Appelé par l'**appel de la classe** : `obj = Rationnel(1,5)`
- Méthode `__init__(self, ...)` dont le premier paramètre est **self**

Exemple (Constructeur)

```
def __init__(self, num, den) :  
    self.__num = num  
    self.__den = den
```



Contexte d'exécution

[toggle](#)[reset](#)

- Une variable = une instance = un contexte d'exécution
- chaque variable contient ses données propres

Exemple

```
x1 = Rationnel(1,2)
x2 = Rationnel(2,3)
print(x1.get_numerateur())
print(x2.get_numerateur())
⇒ 1 2
```



L'objet self

[toggle](#)[reset](#)

- **self** : objet représentant le contexte courant
- utilisable uniquement dans une méthode d'instance
- Utilisable quand on fait appel à une méthode d'instance depuis la classe



L'objet self

[toggle](#)[reset](#)

- **self** : objet représentant le contexte courant
- utilisable uniquement dans une méthode d'instance
- Utilisable quand on fait appel à une méthode d'instance depuis la classe

Exemple (Utilisation de self)

```
class Rationnel(object):  
    def __init__(self, n, d):  
        den = n + d  
        self.__num = n  
        self.__den = d
```



L'objet self

[toggle](#)[reset](#)

- **self** : objet représentant le contexte courant
- utilisable uniquement dans une méthode d'instance
- Utilisable quand on fait appel à une méthode d'instance depuis la classe

Exemple (Utilisation de self)

```
class Rationnel(object):  
    def __init__(self, n, d):  
        den = n + d  
        self.set_num(n)  
        self.__den = d  
    def get_num(self):  
        num = 0  
        return self.__num
```



L'objet self

[toggle](#)[reset](#)

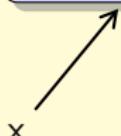
Classe Rationnel

`__num=2`

`__den=3`

`get_num()`

`set_num()`



```
x = Rationnel(2,3)
```



Sommaire

1 Type abstrait de données

Définition

2 Notion d'objet

Concepts fondamentaux

POO en Python

3 Retour sur la notion de TAD

TAD Intervalle

TAD Pile

TAD File

Autres TAD

4 Horloge



Sommaire

1 Type abstrait de données

Définition

2 Notion d'objet

Concepts fondamentaux

POO en Python

3 Retour sur la notion de TAD

TAD Intervalle

TAD Pile

TAD File

Autres TAD

4 Horloge



Intervalle

[toggle](#)[reset](#)

- Principe : défini par des bornes supérieurs et inférieurs.
- Objectif : L'arithmétique d'intervalles a été définie pour estimer des bornes inférieures et supérieures de fonctions non-convexes
- Fonctionalités :
 - Opérations : Addition,, soustraction, multiplication, division,
 $[a,b] + [c,d] = [a+c, b+d]$,
 $[a,b] - [c,d] = [a-d, b-c]$,
 $[a,b] * [c,d] = [\min(ac,ad,bc,bd), \max(ac,ad,bc,bd)]$,
 $[a,b] / [c,d] = [a,b] * [\frac{1}{d}, \frac{1}{c}]$ si $0 \notin [c,d]$.
...
 - Examiner les bornes (inf et sup)



Représentation d'un Intervalle

toggle

reset

Exemple : Intervalle de réelles

- Choix d'une représentation interne :
 - deux réelles
 - représentant une borne supérieure et une borne inférieure
 - etc



Représentation d'un Intervalle

toggle

reset

Exemple : Intervalle de réelles

- Choix d'une représentation interne :
 - deux réelles
 - représentant une borne supérieure et une borne inférieure
 - etc
- **Création d'un Intervalle : création de deux bornes**
- Les opérations et accès aux bornes



Représentation d'un Intervalle

toggle

reset

Exemple : Intervalle ouvert, fermé

- Choix d'une représentation interne :
 - deux réelles
 - **représentant une borne supérieure et une borne inférieure**
 - etc
- Création d'un Intervalle
- Les opérations et accès aux bornes



Intervalle en Python

toggle

reset

- La classe Intervalle

Exemple (Intervalle)

```
class Intervalle(object):
```



Intervalle en Python

toggle

reset

- Le constructeur et les deux variables d'instance

Exemple (Intervalle)

```
class Intervalle(object):
    def __init__(self, mini, maxi):
        self.__inf = mini
        self.__sup = maxi
```



Intervalle en Python

[toggle](#)[reset](#)

- Les décorateurs (accès aux deux bornes)

Exemple (Intervalle)

```
class Intervalle(object):  
    @property  
    def inf(self):  
        """Accès à la borne inférieure.  
        """  
        return self._inf  
    @property  
    def sup(self):  
        """Accès à la borne supérieure  
        """  
        return self._inf
```



Intervalle en Python

toggle

reset

- surcharge des méthodes : ex. la représentation (appelée lors d'une appel de print())

Exemple (Intervalle)

```
class Intervalle(object):
    def __str__(self):
        return '[, ]'.format(self.inf, self.sup)
```



Sommaire

1 Type abstrait de données

Définition

2 Notion d'objet

Concepts fondamentaux

POO en Python

3 Retour sur la notion de TAD

TAD Intervalle

TAD Pile

TAD File

Autres TAD

4 Horloge

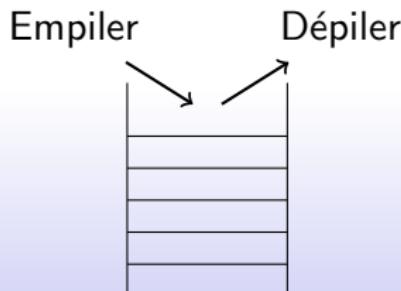


Pile

toggle

reset

- Principe : empiler et dépiler des éléments par le haut de la pile
- Pile = LIFO (Last In First Out)
- Fonctionnalités :
 - Empiler, dépiler
 - Examiner le sommet (sans dépiler)
 - Tester si la pile est pleine ou vide





Représentation de la pile

toggle

reset

Exemple : pile d'entiers

- Choix d'une représentation interne :
 - tableau d'entiers + position du sommet de la pile
 - une liste + capacité autorisée de la pile : possibilité de mettre tout type d'objet
 - etc



Représentation de la pile

toggle

reset

Exemple : pile d'entiers

- Choix d'une représentation interne :
 - tableau d'entiers + position du sommet de la pile
 - une liste + capacité autorisée de la pile : possibilité de mettre tout type d'objet
 - etc
- **Création d'une pile : allouer le tableau**
⇒ voir le polycopié du cours
- Empiler, dépiler, extraire le sommet : accéder au tableau/liste
- Pile vide ou pleine : tester la taille du tableau/capacité de la liste



Représentation de la pile

toggle

reset

Exemple : pile d'éléments

- Choix d'une représentation interne :
 - tableau d'entiers + position du sommet de la pile
 - une liste + capacité autorisée de la pile : possibilité de mettre tout type d'objet
 - etc
- Création d'une pile : définir une liste + capacité
- Empiler, dépiler, extraire le sommet : accéder au tableau/liste
- Pile vide ou pleine : tester la taille du tableau/capacité de la liste



Pile en Python

toggle

reset

Exemple (Pile)

```
class Pile(object):
```



Pile en Python

toggle

reset

Exemple (Pile)

```
class Pile(object):
    def __init__(self, max):
        self.__contenu = []
        self.__maxpile = max
```



Pile en Python

toggle

reset

Exemple (Pile)

```
class Pile(object):

    def estvide(self):
        return len(self.__contenu)==0
    def estpleine(self):
        return len(self.__contenu) == self.__maxpile
```



Pile en Python

toggle

reset

Exemple (Pile)

```
class Pile(object):

    def empiler(self, val):
        if not self.estpleine():
            self.__contenu.append(val)
        else:
            raise ValueError("err : pile pleine")
```



Pile en Python

toggle

reset

Exemple (Pile)

```
class Pile(object):

    def depiler(self):
        if not self.estvide():
            return self.__contenu.pop()
        else:
            return None
```



Pile en Python

toggle

reset

Exemple (Pile)

```
class Pile(object):

    def lire_sommet(self):
        if not self.estvide():
            return self.__contenu[-1]
        return None
```



Utilisation de piles

toggle

reset

- Pile d'appel
- Calcul d'expressions arithmétiques préfixées (ex : $+ 2 * 3 2$)
- Simulation de récursivité en itératif
- Parcours de graphes
- ...



Sommaire

1 Type abstrait de données

Définition

2 Notion d'objet

Concepts fondamentaux

POO en Python

3 Retour sur la notion de TAD

TAD Intervalle

TAD Pile

TAD File

Autres TAD

4 Horloge



TAD File

[toggle](#)[reset](#)

- Principe : Insérer et supprimer des éléments de la file
- File = FIFO (First In First Out)
- Fonctionnalités :
 - Insérer, supprime
 - Examiner la tête (sans défiler)
 - Tester si la file est pleine ou vide





Représentation de la file

toggle

reset

Exemple : file d'entiers

- Représentation interne : liste, tableau + 2 indices
- Création d'une file : initialiser la liste ou le vecteur, allouer le tableau
- Enfiler, défiler, extraire la tête : accéder au tableau
- File vide ou pleine : tester les indices d'entrée et de sortie



Utilisation des files

toggle

reset

- File d'attente
- File de priorité
- Gestion de flux de données
- Parcours de graphes



Sommaire

1 Type abstrait de données

Définition

2 Notion d'objet

Concepts fondamentaux

POO en Python

3 Retour sur la notion de TAD

TAD Intervalle

TAD Pile

TAD File

Autres TAD

4 Horloge



Autres TAD

toggle

reset

- Complexes
- Polynômes
- Arbres
- ...



Sommaire

1 Type abstrait de données

Définition

2 Notion d'objet

Concepts fondamentaux

POO en Python

3 Retour sur la notion de TAD

TAD Intervalle

TAD Pile

TAD File

Autres TAD

4 Horloge



Horloge

toggle

reset

