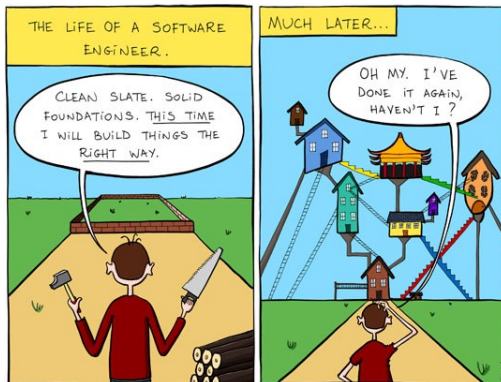


Programmation objet



Les tuples

- Collection ordonnée d'un même type

```
>>> mon_tuple = (3,5,1,3)
```

```
>>> mon_tuple[1]
```

```
5
```

Les tuples

- Collection ordonnée d'un même type

```
>>> mon_tuple = (3,5,1,3)
>>> mon_tuple[1]
5
```

- Un tuple est non mutable

```
>>> mon_tuple[1] = 3
TypeError: 'tuple' object does not support item
assignment
```

Les tuples

- Collection ordonnée d'un même type

```
>>> mon_tuple = (3,5,1,3)
>>> mon_tuple[1]
5
```

- Un tuple est non mutable

```
>>> mon_tuple[1] = 3
TypeError: 'tuple' object does not support item
assignment
```

- Opérations sur les tuples : méthodes count() et index()

```
>>> mon_tuple.count(3)
2
```

Les tableaux

- Collection ordonnée d'un même type

```
>>> mon_tableau = [3,5,1,3]
```

```
>>> mon_tableau[1]
```

```
5
```

Les tableaux

- Collection ordonnée d'un même type

```
>>> mon_tableau = [3,5,1,3]
>>> mon_tableau[1]
5
```

- Un tableau est mutable

```
>>> mon_tableau[1] = 3
>>> mon_tableau
[3,3,1,3]
```

Les tableaux

- Collection ordonnée d'un même type

```
>>> mon_tableau = [3,5,1,3]
>>> mon_tableau[1]
5
```

- Un tableau est mutable

```
>>> mon_tableau[1] = 3
>>> mon_tableau
[3,3,1,3]
```

- Opérations sur les tableaux

```
>>> mon_tableau.append(7)
>>> mon_tableau
[3,3,1,3,7]
```

Structure Chrono

On peut ajouter des structures de données en définissant une *classe*.

Structure Chrono

On peut ajouter des structures de données en définissant une *classe*.

Exemple : manipuler des triplets d'entier représentant le temps en heures, minutes, secondes.

On peut définir une structure Chrono :



Chrono
heures
minutes
secondes


Classe Chrono avec Python

mot clef class
suivi du nom et :

`class Chrono:`

*'''une classe pour representer un temps mesure
en heures, minutes et secondes'''*

`def __init__(self, h, m, s):`   **fonction __init__**

`self.heures = h` 

`self.minutes = m`

`self.secondes = s`

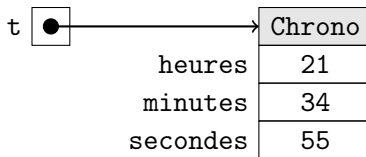
**affectation
des valeurs
aux attributs**

Création d'un objet

On peut affecter à la variable `t` un *objet* représentant le temps «21 heures, 34 minutes et 55 secondes» de la manière suivante :

```
>>> t = Chrono(21,34,55)
```

On obtient la situation suivante :



Manipulation des attributs

On peut consulter la valeur d'un attribut :

```
>>> t.heures  
21
```

Manipulation des attributs

On peut consulter la valeur d'un attribut :

```
>>> t.heures  
21
```

En Python, on peut également modifier la valeur d'un attribut :

```
>>> t.heures += 1  
>>> t.heures  
22
```

Les valeurs des attributs est parfois appelé *état* de l'objet.

Attributs de classe

Un attribut de classe est une valeur attachée à la classe elle-même.

```
class Chrono:  
    heure_max = 24  
    ...
```

```
>>> t = Chrono(21, 34, 55)  
>>> (t.heure_max, Chrono.heure_max)  
(24, 24)
```

Méthodes

Les fonctions qui permettent de manipuler un objet sont appelées *méthodes*.

Méthodes

Les fonctions qui permettent de manipuler un objet sont appelées *méthodes*.

Méthode `texte` qui affiche une chaîne de caractère décrivant le temps représenté :

```
>>> t.texte()  
21h 34m 55s
```


Méthodes

Les fonctions qui permettent de manipuler un objet sont appelées *méthodes*.

Méthode `texte` qui affiche une chaîne de caractère décrivant le temps représenté :

```
>>> t.texte()  
21h 34m 55s
```

Méthode `avance` faisant avancer d'un certain nombre de secondes :

```
>>> t.avance(5)  
>>> t.texte()  
21h 35m 00s
```

Définir une méthode

```
def texte(self):  
    return (str(self.heures)+'h'  
            +str(self.minutes)+'m'  
            +str(self.secondes)+'s')
```

Définir une méthode

```
def texte(self):  
    return (str(self.heures)+'h'  
            +str(self.minutes)+'m'  
            +str(self.secondes)+'s')  
  
def avance(self,s):  
    self.secondes += s  
    # depassement secondes  
    self.minutes += self.secondes//60  
    self.secondes = self.secondes%60  
    #depassement minutes  
    self.heures += self.minutes//60  
    self.minutes = self.minutes%60
```

Constructeur

```
t = Chrono(21, 34, 55)
```

- 1 création de l'objet lui-même
- 2 appel à une méthode spéciale, appelée *constructeur*, chargé d'initialiser les valeurs des attributs. En Python, il s'agit de la méthode `__init__`.

Autres méthodes particulières en Python

méthode	appel	effet
<code>__str__(self)</code>	<code>str(t)</code> <code>print(t)</code>	renvoie une chaîne de caractère décrivant <code>t</code>
<code>__lt__(self,u)</code>	<code>t < u</code>	revoie True si <code>t</code> est strictement plus petit que <code>u</code>
<code>__len__(self)</code>	<code>len(t)</code>	renvoie un nombre entier définissant la taille de <code>t</code>
<code>__contains__</code>	<code>x in t</code>	revoie True si <code>x</code> est dans la collection <code>t</code>
<code>__getitem__(self,i)</code>	<code>t[i]</code>	renvoie le <code>i</code> -ième élément de <code>t</code>

Autres méthodes particulières en Python

méthode	appel	effet
<code>__str__(self)</code>	<code>str(t)</code> <code>print(t)</code>	renvoie une chaîne de caractère décrivant t
<code>__lt__(self,u)</code>	<code>t < u</code>	revoie True si t est strictement plus petit que u
<code>__len__(self)</code>	<code>len(t)</code>	renvoie un nombre entier définissant la taille de t
<code>__contains__</code>	<code>x in t</code>	revoie True si x est dans la collection t
<code>__getitem__(self,i)</code>	<code>t[i]</code>	renvoie le i-ième élément de t

```
def __str__(self):  
    return self.texte()
```

Public et Privé

- **Public** : accessible depuis n'importe quelle partie du code

Public et Privé

- **Public** : accessible depuis n'importe quelle partie du code
- **Privée** : uniquement accessible depuis l'objet lui-même

Public et Privé

- **Public** : accessible depuis n'importe quelle partie du code
- **Privée** : uniquement accessible depuis l'objet lui-même
- Pas d'information privées en Python mais une convention : ce qui commence par un souligné « `_` » ne devrait pas être utilisé.

Public et Privé

- **Public** : accessible depuis n'importe quelle partie du code
- **Privée** : uniquement accessible depuis l'objet lui-même
- Pas d'information privées en Python mais une convention : ce qui commence par un souligné « `_` » ne devrait pas être utilisé.

Public et Privé

- **Public** : accessible depuis n'importe quelle partie du code
- **Privée** : uniquement accessible depuis l'objet lui-même
- Pas d'information privées en Python mais une convention : ce qui commence par un souligné « `_` » ne devrait pas être utilisé.

Chrono	
<code>_heures</code>	<code>__init__</code>
<code>_minutes</code>	<code>texte</code>
<code>_secondes</code>	<code>avance</code>

Modification de la classe Chrono

Simplifier les opérations à l'aide d'un attribut `_temps` :

```
class Chrono:  
    def __init__(self, h, m, s):  
        self._temps = 3600*h + 60*m + s
```

Modification de la classe Chrono

Simplifier les opérations à l'aide d'un attribut `_temps` :

```
class Chrono:
    def __init__(self, h, m, s):
        self._temps = 3600*h + 60*m + s

    def avance(self, s):
        self._temps += s
```

Modification de la classe Chrono

Simplifier les opérations à l'aide d'un attribut `_temps` :

```
class Chrono:
    def __init__(self, h, m, s):
        self._temps = 3600*h + 60*m + s

    def avance(self, s):
        self._temps += s

    def texte(self):
        h, m, s = self._conversion()
        return str(h)+'h '+str(m)+'m '+str(s)+'s'
```

```
def _conversion(self):  
    s = self._temps  
    return (s//3600, (s//60)%60, s%60)
```

```
def _conversion(self):  
    s = self._temps  
    return (s//3600, (s//60)%60, s%60)
```

Chrono	
_temps	<code>--init--</code> texte avance _conversion

L'*interface* n'a pas changé.