CLASSE GODOT

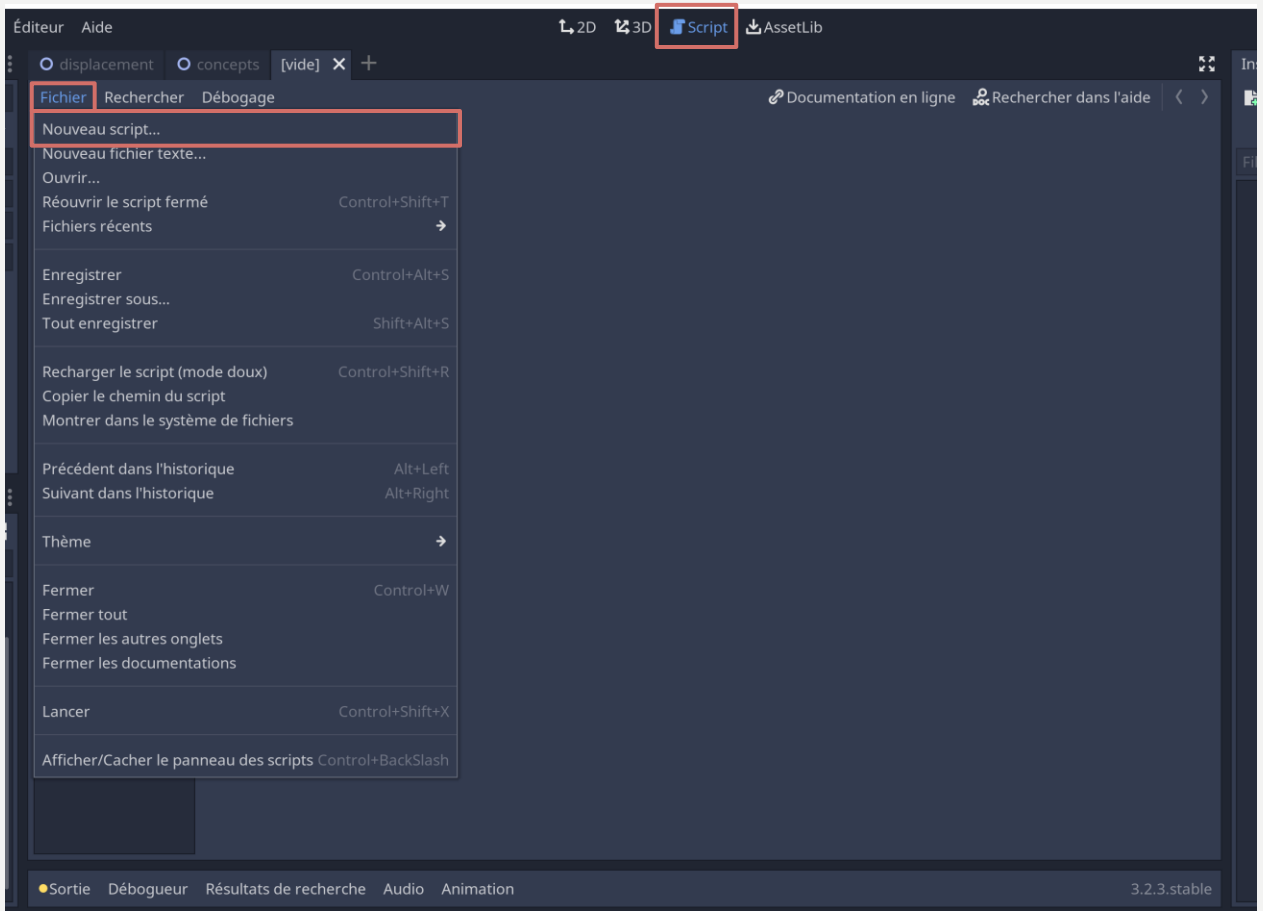
CRÉATION ET UTILISATION

INTRODUCTION

- Dans Godot, quasiment tout est objet. Un *Node2D* est un objet, un *Sprite2D* est un objet, …
- Il faut savoir que l'on peut créer nos propres objets si on en a besoin.
- Il y a plusieurs raisons à vouloir créer nos propres objets :
  - On peut créer un objet spécifique que Godot ne propose pas. Par exemple pour un jeu au moyen âge, on veut par exemple un objet Arme, un objet Casque, un objet Gant, …
  - On peut créer un objet pour donner du sens à ce qu'on manipule. Par exemple, un *Vector2*, on se sait pas si on manipule une position, un déplacement, une vitesse, une accélération, …
  - Etendre un objet déjà existant qui ne propose pas tout ce que l'on souhaite. Exemple, dans Godot, il existe les objets *Sprite2D* qui affiche une image. On veut pouvoir avoir un objet comme *Sprite2D*, mais qui fait des choses en plus comme afficher une autre image lorsque la souris passe dessus.

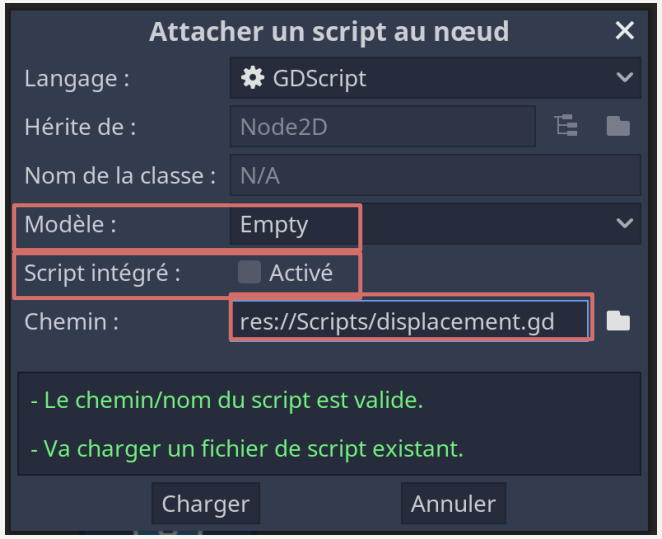
CRÉATION D'UN OBJET SPÉCIFIQUE

- Pour créer un nouvel objet avec Godot, il faut d'abord créer un script standalone:
  - Créer un nouveau script. Il faut aller dans l'onglet Script et dans le menu Fichier, cliquer sur « Nouveau script »



CRÉATION D'UN OBJET SPÉCIFIQUE

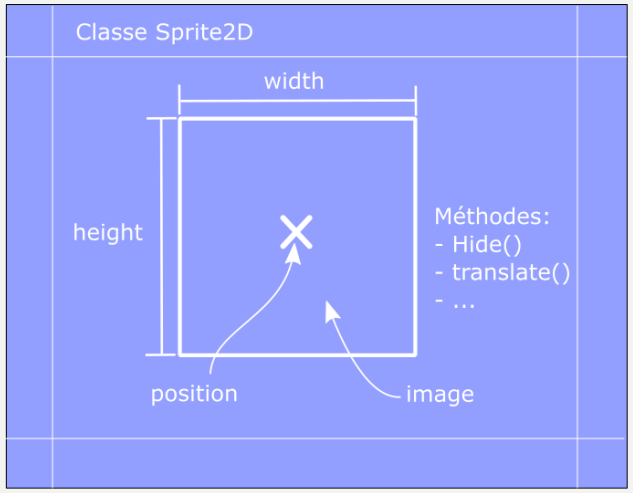
- Une nouvelle pop-up s'affiche pour la création d'un script
- Dans les options, il faut vérifier que l'option « Script intégré » est bien désactivée (sinon le script va être attaché à un nœud)
- Mettre le modèle à Empty, on a pas besoin de script pré-rempli
- Il faut bien enregistrer le script dans le dossier Scripts (histoire que notre dossier de projet soit propre)
- Cliquer sur charger pour finaliser la création du script



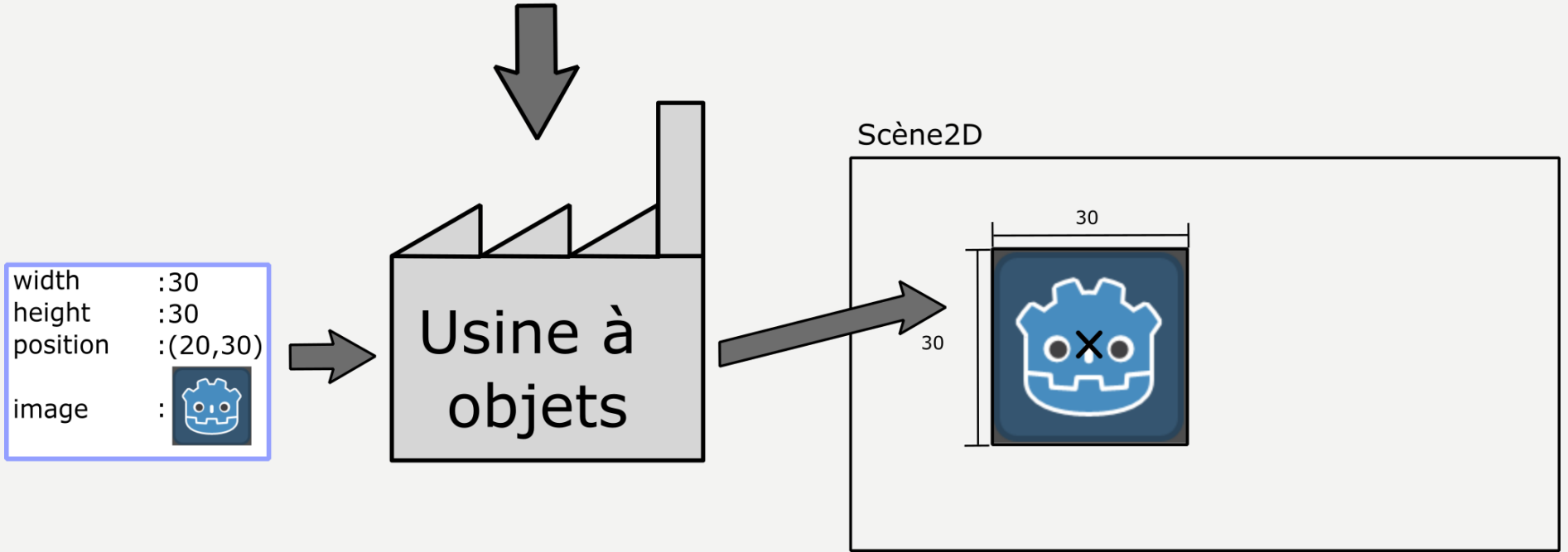
CRÉATION D'UN OBJET SPÉCIFIQUE

- Dans le nouveau script qui vient de s'ouvrir, on va pouvoir commencer la création d'une nouvelle classe.
- Alors oui, on a parlé que d'objets jusqu'à maintenant, et maintenant on parle de classe, c'est quoi une classe et c'est quoi la différence entre une classe et un objet?
- Petite explication slide suivante

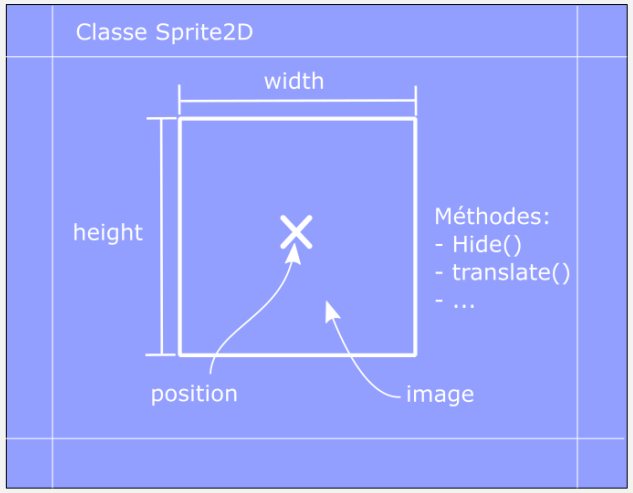
CRÉATION D'UN OBJET SPÉCIFIQUE



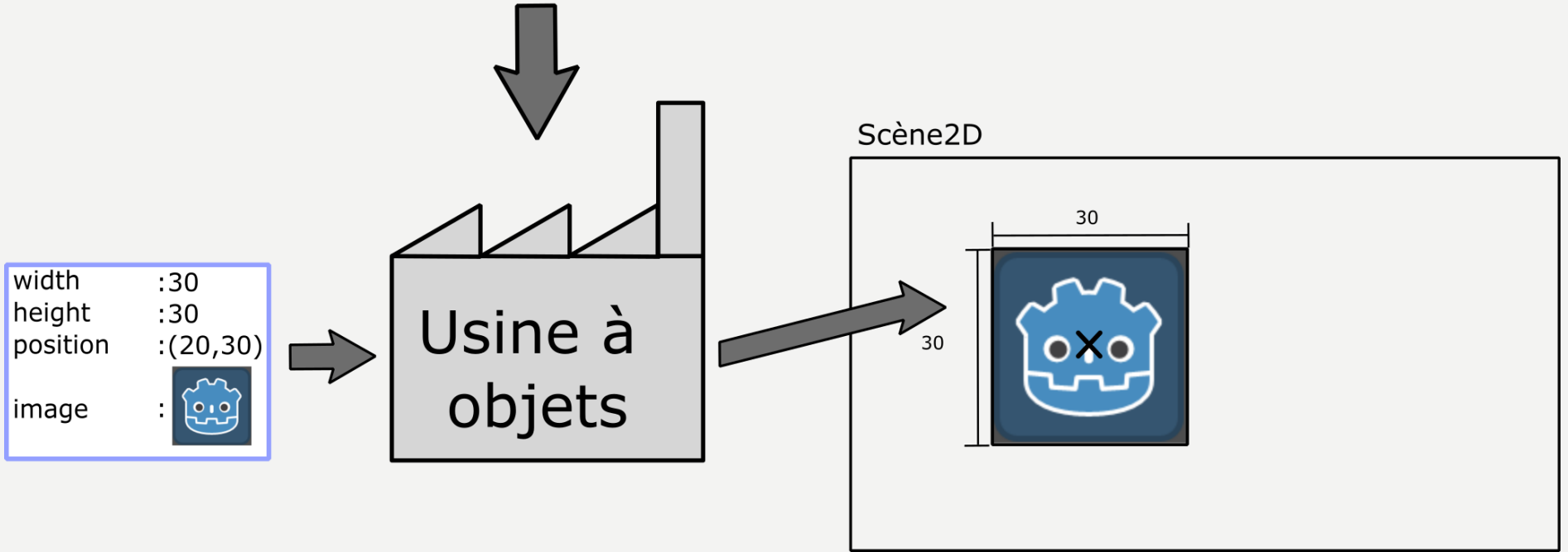
Il faut voir une classe comme un plan de fabrication pour un certain type d'objet.
Dans l'exemple ici, on représente un plan de fabrication d'un *Sprite2D*.
Ce *Sprite2D* a des attributs comme une hauteur, une largeur, … et des méthodes pour manipuler ces attributs.



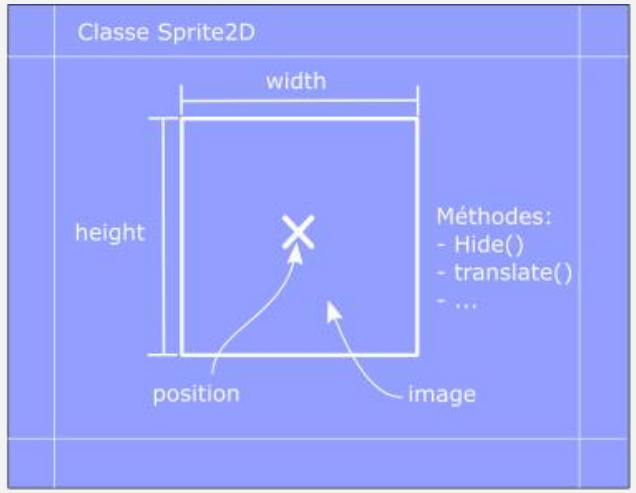
CRÉATION D'UN OBJET SPÉCIFIQUE



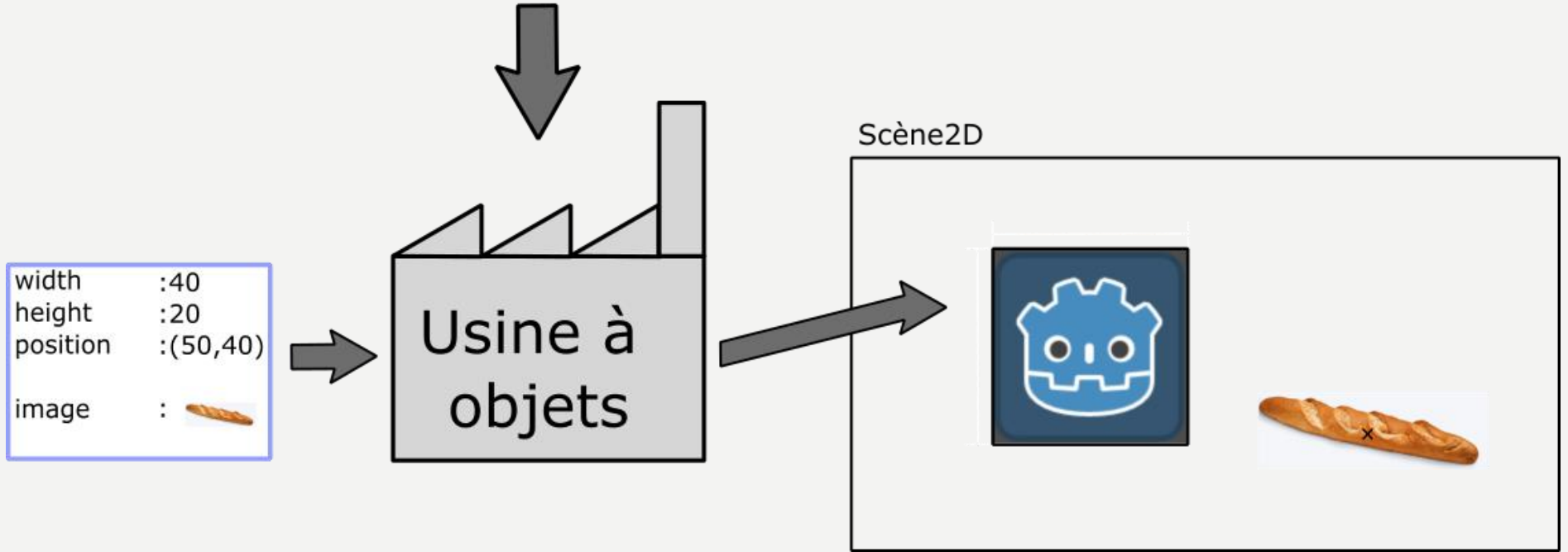
Mais avec ce plan on a pas d'objets, il nous faut une usine et ce qu'on souhaite comme valeur pour la taille du sprite, quelle image on veut mettre, à quelle position afin de créer un nouvel objet.
Ici par exemple en veut créer un nouveau *Sprite2D* dans la scène qui a une taille 30x30, une position (20,30) et une image du Godot.



CRÉATION D'UN OBJET SPÉCIFIQUE



Avec ce même plan, je peux tout à fait créer un autre sprite avec des valeurs différentes.
Par exemple : une taille de 40x20, une position (50, 40) et une image d'une baguette



CRÉATION D'UN OBJET SPÉCIFIQUE

- Donc! Créons une nouvelle classe. Cette classe va s'appeler **Displacement** et va avoir pour but de donner plus de sens à l'utilisation de *Vector2*.
- Voici un exemple:

```
1  class_name Displacement
2
3  var value : Vector2
4
5  var dx setget dx_set, dx_get
6
7  func dx_set(new_value):
8      value.x = new_value
9
10 func dx_get():
11     return value.x
12
13 func add_displacement(var disp : Displacement):
14     value.x += disp.dx
15     value.y += disp.dy
16
17 func _init(var dx, var dy):
18     value.x = dx
19     value.y = dy
```

CRÉATION D'UN OBJET SPÉCIFIQUE

- Essayons de décortiquer

```
1  class_name Displacement
```

- class_name permet de donner un nom à la classe, ici **Displacement**

```
3  var value : Vector2
```

- Ici un attribut de la classe est déclarée. On utilise un *Vector2*, puisque c'est à lui que l'on veut donner plus de sens.

CRÉATION D'UN OBJET SPÉCIFIQUE

```
17 func _init(var dx, var dy):
18     value.x = dx
19     value.y = dy
```

- Cette méthode est le constructeur de la classe **Displacement**
- Il permet de créer un objet **Displacement** en affectant des nouvelles valeurs
- Attention : il faut absolument que le nom du constructeur s'appelle *_init*, sinon il ne sera pas appelé lors de la création de l'objet

```
func _ready():
    var displacement = Displacement.new(5,3)
```

- Exemple : on créé un nouveau **Displacement** grâce à la méthode *new* qui va appeler notre constructeur *_init* en envoyant les valeurs 5 pour *dx* et 3 pour *dy*
- Pourquoi on fait pas *Displacement._init(5,3)*? Parce que la méthode new fait pas mal de choses dont l'appel de notre méthode *_init*.
- La méthode *new* doit absolument être utilisée pour créer un nouvel objet **Displacement**

CRÉATION D'UN OBJET SPÉCIFIQUE

```
5   var dx setget dx_set, dx_get
6
7   func dx_set(new_value):
8       value.x = new_value
9
10  func dx_get():
11      return value.x
```

- Ici, cette étape est importante, puisqu'on utilise une technique qui permet de donner un surnom à l'attribut « *x* » de *Vector2*. On va appeler ce surnom « *dx* ».
- Afin de modifier la valeur *x* de l'attribut value, on va donc écrire une fonction *dx_set* prenant comme paramètre d'entrée la nouvelle valeur à affecter
- Afin de récupérer la valeur *x* de l'attribut value, on va donc écrire une fonction *dx_get* retournant la valeur courant de *x*
- Par contre c'est pas très pratique d'écrire à chaque fois :
  displacement.set_dx(4) , displacement.get_dx(), displacement.set_dx(2), …
- On veut plutôt écrire de la façon suivante:
  displacement.dx = 4 , displacement.dx , displacement.dx = 2, …

CRÉATION D'UN OBJET SPÉCIFIQUE

```
5   var dx setget dx_set, dx_get
6
7   func dx_set(new_value):
8       value.x = new_value
9
10  func dx_get():
11      return value.x
```

- C'est pour cela que l'on voit *setget* juste après la déclaration de *dx* avec les noms *dx_set* et *dx_get*
- Ce que çà signifie c'est « Pour la variable *dx* je souhaite que lorsque je fais *disp.dx = 5*, cela appelle implitement *disp.dx_set(5)* et quand je fais *var ma_variable = disp.dx*, cela appelle implicitement *ma_variable = disp.dx_get()*
- La structure est alors la suivante :

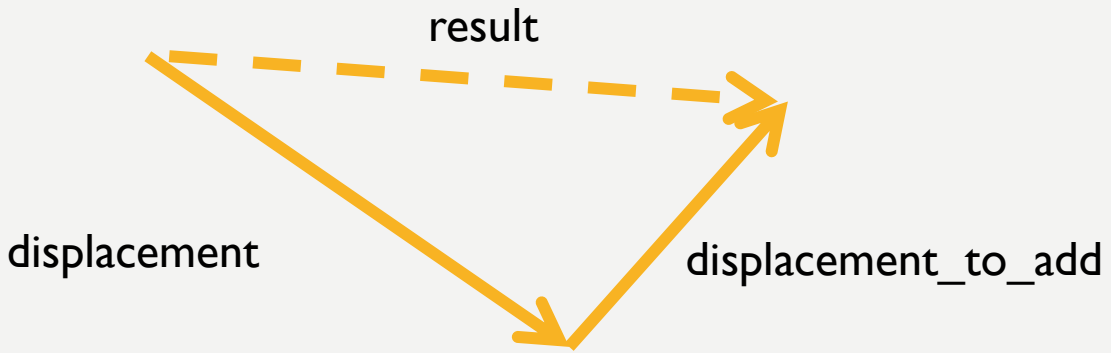
```
var nom_variable setget nom_fonction_set nom_fonction_get

func nom_fonction_set
```

CRÉATION D'UN OBJET SPÉCIFIQUE

```
13 func add_displacement(var disp : Displacement):
14     value.x += disp.dx
15     value.y += disp.dy
```

- Cette méthode permet d'ajouter un autre déplacement au déplacement actuel



- Imaginons que notre déplacement actuel est displacement et que l'on souhaite ajouter un autre déplacement displacement_to_add. Le résultat de l'addition des deux déplacement donnera le déplacement result.

CRÉATION D'UN OBJET SPÉCIFIQUE

- On peut ajouter une dernière fonction à notre classe **Displacement** afin de pouvoir faire ce genre de chose : `print(displacement)`
- En effet, si on ne créé pas de fonction spéciale pour dire quoi afficher dans la classe **Displacement**, `print(displacement)` affichera non pas le contenu de la classe **Displacement**, mais simplement son identifiant
- La fonction spéciale qu'il faut écrire est la suivante:

```
28 func _to_string() -> String:
29     return "(" + String(value.x) + "," + String(value.y) +")"
```

- Comme tu peux le remarquer, on a le même « _ » avant le nom, comme pour la fonction *_init*, *_ready* ou *_process*. Ce sont des fonctions qui existent déjà, mais que l'on va réécrire à notre sauce. C'est la notion d'héritage, mais on reviendra dessus plus tard. Il faut juste savoir qu'il faut bien écrire la signature de la fonction comme çà.
- Pour la valeur de retour, on a fait en sorte de retourner une chaine de caractère contenant les deux valeurs *x* et *y* de notre classe.

UTILISATION DE LA CLASSE

- Voici un exemple d'utilisation de la classe **Displacement**:

```
func _ready():
	var displacement = Displacement.new(5,3)
	print(displacement) # affiche (5,3)

	var displacement_to_add = Displacement.new(2,2)
	displacement.add_displacement(displacement_to_add)
	print(displacement) # affiche (7,5)

	displacement.dx = 6
	displacement.dy = 3
	print(displacement) # affiche (6,3)
```

EXERCICE

- Saurais-tu réécrire la classe **Displacement** de ton côté sans regarder?
- Ecris une nouvelle classe appelée Position qui contient :
  - Un attribut de type *Vector2*
  - Deux getset appelés *x* et *y* pour l'attribut *x* et *y* du *Vector2*
  - Un constructeur permettant d'initialiser *x* et *y*
  - Une méthode appelée *move*, prenant en paramètre une variable de type **Displacement** et qui va ajouter le déplacement à la position courante
  - Une méthode *_to_string()* permettant de retourner une chaine de caractères affichant les valeurs *x* et *y* de l'attribut
- Fais un petit script de test qui permet :
  - d'instancier une position *pos1* avec comme valeur (4,3), un déplacement *disp1* avec comme valeur (3,8)
  - D'appliquer le déplacement *disp1* sur *pos1*
  - D'afficher *pos1*