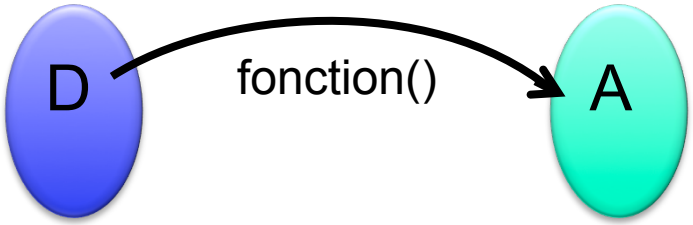
Chapitre 5 : aspects fonctionnels

Langage Fonctionnel

- Vision mathématique de la notion de fonction :
  - Associe à **l'ensemble de départ**, une seule valeur dans **l'ensemble d'arrivée**



  - Pas d'effet de bord : pour les mêmes valeurs en entrées toujours la même valeur rendue.
  - La fonction peut être "tabulée" si le nombre d'entrées est fini et pas trop grand
  - **Ensemble de départ** : paramètres de la fonction
  - **Ensemble d'arrivée** : type du résultat de la fonction
- La fonction est aussi un type d'entité manipulable explicitement
  - on peut stocker une fonction dans une variable
  - retourner une fonction comme résultat d'une fonction !

Fonctionnel : appel par indirection de variable string

```
// Fonction nommée classique
function add1(int $x) : int {
  return $x +1;
}

// Stocke cette fonction dans une variable (juste son nom)
$fonction = 'add1';

$a = 10;

// appelle la fonction dont le nom est dans une variable
$b = $fonction($a);
```

Fonction anonyme de type callable

```php
// Fonction anonyme dans une variable
$maFonction = function (int $x) : int {
  return $x +1;
};

// appelle cette Fonction
$a = 10;
$b = $maFonction($a);
```

- La valeur n'est plus une string mais une valeur de type **callable**

Fonction anonyme en paramètre d'une fonction

```
// Fait la même chose sur tous les elements d'un tableau
function map(array $tab, callable $do) {
  foreach ($tab as $key => $value) {
        // applique la fonction à un element
        $tab[$key] = $do($tab[$key]);
  }
  return $tab;
}

$notes = array (3,6,12,16,2,1);

// Force toutes les notes à 8 minimum
$notes = map($notes,function(int $i){
                        if ($i < 8) {return 8;};
                        return $i;}
  );
```

Retour d'une fonction anonyme et closure

```
// Produit un fonction avec un seuil
function seuillage(int $seuil) : callable {
  // retourne une fonction de seuil avec closure sur $seuil
  return function(int $i) use($seuil) {
      if ($i < $seuil)
        {return $seuil;};
      return $i;};
}

$notes = array (3,6,12,16,2,1);

// Force toutes les notes à 7 minimum
$notes = map( $notes, seuillage(7));
```

Application au tri : ordre sur les objets

```
class Fruit {
  function __construct(string $nom) {
    $this->nom = $nom;
  }
}

$tab[] = new Fruit("pomme");;
$tab[] = new Fruit("bannane");;
$tab[] = new Fruit("ananas");
$tab[] = new Fruit("poire");
```

- Que fait : sort($tab) ?

  Utilise une comparaison implicite entre objets : il vaut mieux la rendre explicite !

Paramètre fonctionnel string

```
class Fruit {
        public $nom;
        public $prix;
...

// Fonction de comparaison
function compNom_Fruit(Fruit $a, Fruit $b) : bool {
  return $a->nom < $b->nom;
}
// tri avec choix de l'ordre par la fonction
usort($tab,'compNom_Fruit');
```

Fonction : bool usort (array &$array , callable $value_compare_func)

- Choisir sur quoi on va faire le tri
- Passe la fonction en paramètre du tri

Paramètre fonctionnel callable

```
class Fruit {
        public $nom;
        public $prix;
...

// Fonction de comparaison
$compNom_Fruit = function (Fruit $a, Fruit $b) : bool {
  return $a->nom < $b->nom;
}
// tri avec choix de l'ordre par la fonction
usort($tab,$compNom_Fruit);
```

Fonction qui retourne une fonction avec closure

```
class Fruit {
        public $nom;
        public $prix;
...

// Retourne une fonction avec closure sur $attribut
function ordonner_sur(string $attribut) : callable {
    return function ($a, $b) use ($attribut) {
        return $a->$attribut > $b->$attribut;
    };
}

// tri sur le nom
usort($tab, ordonner_sur('nom'));

// tri sur le prix
usort($tab, ordonner_sur('prix'));
```

Conclusion sur les aspects fonctionnels

- PHP est aussi un langage fonctionnel avec le type **callable**
  - Nom de fonction dans une variable
  - Fonction anonyme (pas de nom)
- Possède la notion de fonctionnelle de fermeture (closure)
  - Même notion que dans Javascript (et C++, Java)
  - Mais il faut explicitement indiquer les variables de la fermeture (**use**)
- Usage similaire à de la généricité
- Utile dans des cas particulier comme l'ordre des tris
- En général : préférer les algorithmes fonctionnel aux impératifs
  - Plus de puissance d'expression
  - Plus compact dans le code
  - Pas d'effets de bords sur les variables
  - Efficacité ? dépend de l'interpréteur (ou compilateur pour C++)