Didactique de l'informatique

M1 MEEF NSI

Variété des langages et environnements

Emmanuel Beffara

INSPÉ Institut national supérieur du professorat et de l'éducation Académie de Grenoble

2022 – 2023

Plan

Au collège

L'informatique dans le programme de cycle 4

`https://eduscol.education.fr/90/j-enseigne-au-cycle-4`

- en maths (p. 137), c'est succinct
- en techno (pp. 123–125) c'est ambitieux

Repères et attendus en maths

`https://eduscol.education.fr/137/attendus-de-fin-d-annee-et-reperes-annuels-de-progression-du-cp-la-3e`

- c'est trois fois la même chose pour algorithmique et progammation, sauf le niveau d'attendus (1 à 3)
- au niveau 3, il faut savoir accumuler des résultats dans des variables, utiliser boucles, évènements et blocs personnalisés

C'est très orienté vers Scratch (sans le dire).

En seconde

Programme de maths de 2nde générale

https://www.education.gouv.fr/bo/19/Special1/MENE1901631A.htm?cid_bo=138131

- un peu d'algorithmique partout
- le gros morceau : la notion de fonction

Programme de SNT

https://www.education.gouv.fr/bo/19/Special1/MENE1901641A.htm?cid_bo=138143

- notions transversales de programmation : cycle 4 + fonctions
- référence explicite au cours de maths

Après : en maths, principalement la notion de liste.

Plan

Similitudes

Scratch et Python sont tous les deux des langages :

- séquentiels ;
- impératifs ;
- interprétés.

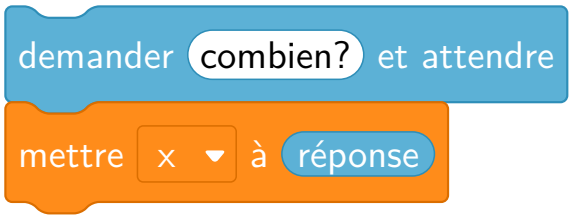
Dictionnaire bilingue : premiers éléments

début du programme


```
x = valeur
```


```
print("bonjour")
```


```
x += valeur
```

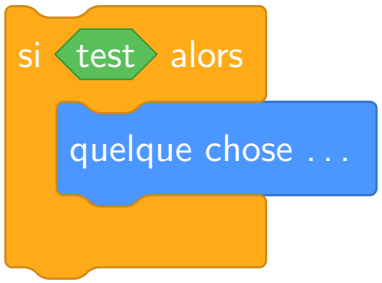



```
x = input("combien?")
```

Différences

Python	**Scratch**
les valeurs ont un type précis (entier, flottant, chaîne...)	pas de notion de type et des conversions implicites

Dictionnaire bilingue : conditionnelles




```
if test:
    # quelque chose...
```


```
if test:
    # c'est vrai...
else:
    # c'est faux...
```

Similitudes

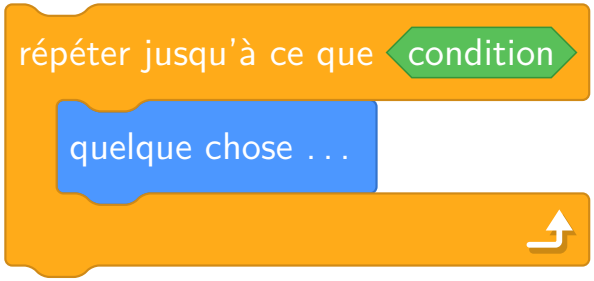
Scratch et Python sont tous les deux des langages :

- séquentiels ;
- impératifs ;
- interprétés.

En plus :

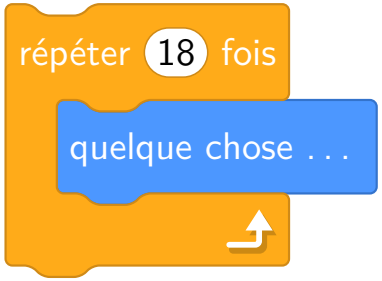
- la séquentialité = le passage à la ligne suivante ;
- l'indentation en Python colle bien avec la présentation des sous-blocs en Scratch ;

Dictionnaire bilingue : boucles



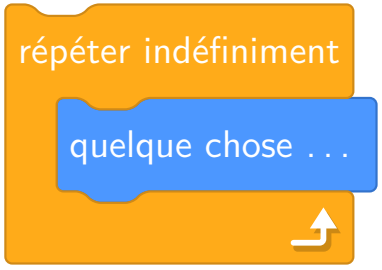

```
while not condition:
    # quelque chose...
```

La condition est inversée




```
for i in range(18):
    # quelque chose...
```

Pas d'indice côté Scratch




```
while True:
    # quelque chose...
```

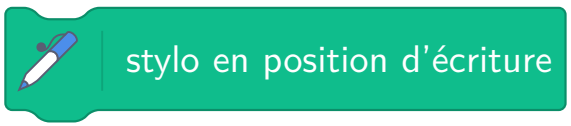
Besoin d'une condition triviale

Différences

Python	**Scratch**
les valeurs ont un type précis (entier, flottant, chaîne...)	pas de notion de type et des conversions implicites
les boucles bornées sont des parcours de structures	la boucle bornée ne donne pas accès au rang d'itération

Dictionnaire bilingue : dessiner

Avec la bibliothèque `turtle` de Python.




```
pendown()
```


```
penup()
```


```
forward(10)
```

Un outil de transition ?

Dessin : exemples classiques

- frises
- carrés concentriques
- ...

Une analyse didactique de ces exercices ?

Boucles et variables

Le cas des boucles bornées est particulièrement délicat :

- Scratch est artificiellement bridé (pas d'accès au rang de l'itération).
- Python utilise un mécanisme d'itération très général

```
for x in seq:
    ...
```

- En algorithmique classique, on a tendance à beaucoup insister sur
  pour i allant de 1 *à n . . .*

  ce qui n'est naturel ni en Scratch ni en Python.
- C'est un point explicite du programme.

Dictionnaire bilingue : sous-programmes


```
def manipuler(x):
    # faire quelque chose avec x...
```


```
manipuler(42)
```

Similitudes

Scratch et Python sont tous les deux des langages :

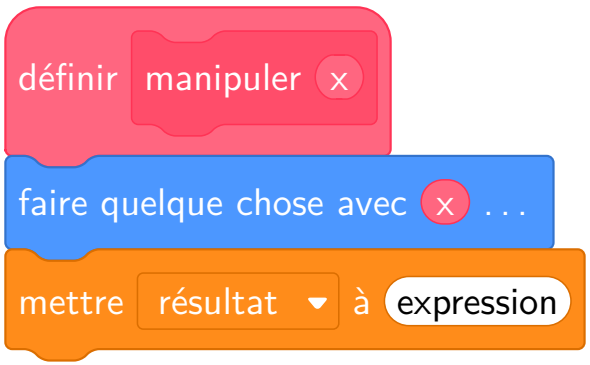
- séquentiels ;
- impératifs ;
- interprétés.

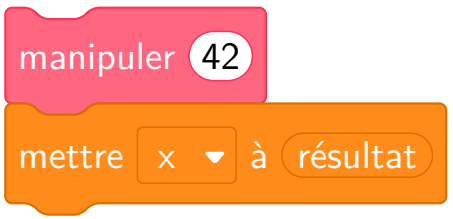
En plus :

- la séquentialité = le passage à la ligne suivante ;
- l'indentation en Python colle bien avec la présentation des sous-blocs en Scratch ;
- les blocs personnalisés permettent d'introduire la notion de fonction

Dictionnaire bilingue : sous-programmes

Et pour mettre une valeur de retour ?






```
x = manipuler(42)
```

```
def manipuler(x):
    # faire quelque chose avec x...
    return expression
```

C'est un peu du bricolage.

Différences

Python	**Scratch**
les valeurs ont un type précis (entier, flottant, chaîne...)	pas de notion de type et des conversions implicites
les boucles bornées sont des parcours de structures	la boucle bornée ne donne pas accès au rang d'itération
les appels de fonctions sont des expressions	les blocs personnalisés définissent des instructions

Fonctions

C'est le point majeur du programme de seconde :

- c'est une notion nouvelle et difficile ;
- c'est la manière officiellement recommandée de présenter les algorithmes ;
- le lien avec la notion de fonction en maths est à exploiter et à éclaircir ;
- c'est le lieu où doivent se discuter des concepts comme la portée des variables ou l'ordre d'évaluation des expressions ;
- en Python, une fonction est un objet comme les autres (on peut le passer en argument).

La programmation évènementielle multi-agents

Des éléments typiques de Scratch :

- programmation objet
- programmation évènementielle

Similitudes

Scratch et Python sont tous les deux des langages :

- ~~séquentiels~~ ; *Scratch, pas vraiment*
- impératifs ;
- interprétés.

En plus :

- la séquentialité = le passage à la ligne suivante ;
- l'indentation en Python colle bien avec la présentation des sous-blocs en Scratch ;
- les blocs personnalisés permettent d'introduire la notion de ~~fonction~~ procédure.

Différences

Python	**Scratch**
les valeurs ont un type précis (entier, flottant, chaîne...)	pas de notion de type et des conversions implicites
les boucles bornées sont des parcours de structures	la boucle bornée ne donne pas accès au rang d'itération
les appels de fonctions sont des expressions	les blocs personnalisés définissent des instructions
on lance *un* programme	les lutins suivent *des* scripts et interagissent

L'écosystème

Une cible différente :

- Scratch a été conçu pour l'éducation ;
- Python est un langage généraliste.

Conséquences :

- Scratch est prêt pour l'algorithmique au collège ;
- Python est prêt pour l'informatique au lycée ;
- Python peut être étendu par de nombreuses bibliothèques pour à peu près tout.

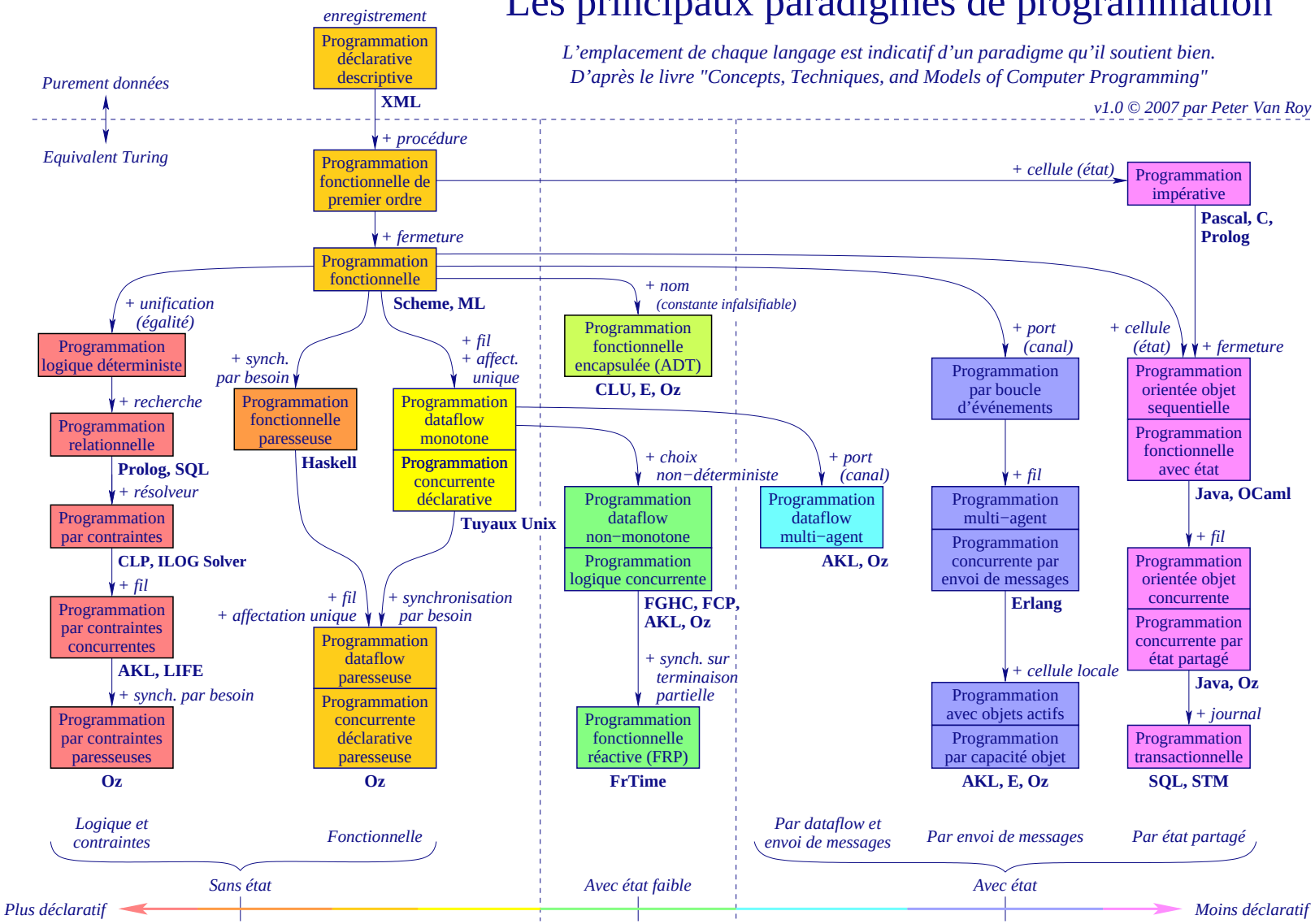
Gérer la transition

Comment construire les premières séances ?

- Identifier les acquis de collège.
- Déterminer un objectif en accord avec le programme de seconde.
- Construire une situation en conséquence.

Plan

Paradigmes...


Les principaux paradigmes de programmation
L'emplacement de chaque langage est indicatif d'un paradigme qu'il soutient bien.
D'après le livre "Concepts, Techniques, and Models of Computer Programming"
v1.0 © 2007 par Peter Van Roy
Purement données
Equivalent Turing
enregistrement
Programmation déclarative descriptive
XML
+ procédure
Programmation fonctionnelle de premier ordre
+ cellule (état)
Programmation impérative
Pascal, C, Prolog
+ fermeture
Programmation fonctionnelle
Scheme, ML
+ unification (égalité)
+ nom (constante infalsifiable)
+ port (canal)
+ cellule (état)
+ fermeture
Programmation logique déterministe
+ synch. par besoin
+ fil + affect. unique
Programmation fonctionnelle encapsulée (ADT)
CLU, E, Oz
Programmation par boucle d'événements
Programmation orientée objet sequentielle
Programmation fonctionnelle avec état
+ recherche
Programmation fonctionnelle paresseuse
Haskell
Programmation dataflow monotone
Programmation concurrente déclarative
Tuyaux Unix
+ choix non−déterministe
+ port (canal)
Programmation relationnelle
Prolog, SQL
+ résolveur
Programmation dataflow non−monotone
Programmation logique concurrente
Programmation dataflow multi−agent
AKL, Oz
+ fil
Programmation multi−agent
Programmation concurrente par envoi de messages
Erlang
Java, OCaml
+ fil
Programmation par contraintes
CLP, ILOG Solver
+ fil
+ affectation unique
+ synchronisation par besoin
FGHC, FCP, AKL, Oz
Programmation orientée objet concurrente
Programmation concurrente par état partagé
Java, Oz
Programmation par contraintes concurrentes
AKL, LIFE
+ synch. par besoin
Programmation dataflow paresseuse
Programmation concurrente déclarative paresseuse
+ synch. sur terminaison partielle
+ cellule locale
+ journal
Programmation par contraintes paresseuses
Oz
Oz
Programmation fonctionnelle réactive (FRP)
FrTime
Programmation avec objets actifs
Programmation par capacité objet
AKL, E, Oz
Programmation transactionnelle
SQL, STM
Logique et contraintes
Fonctionnelle
Par dataflow et envoi de messages
Par envoi de messages
Par état partagé
Sans état
Avec état faible
Avec état
Plus déclaratif
Moins déclaratif

Différents paramètres

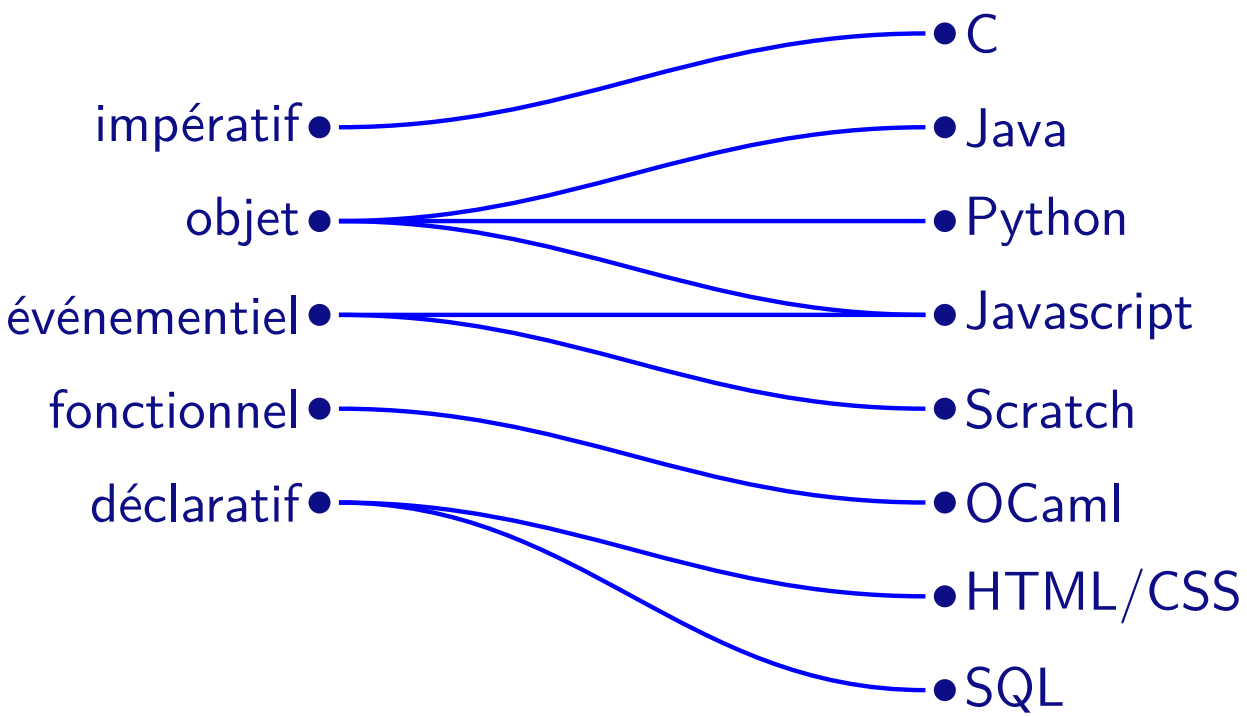
- impératif
- objet
- événementiel
- fonctionnel
- déclaratif

Différents paramètres

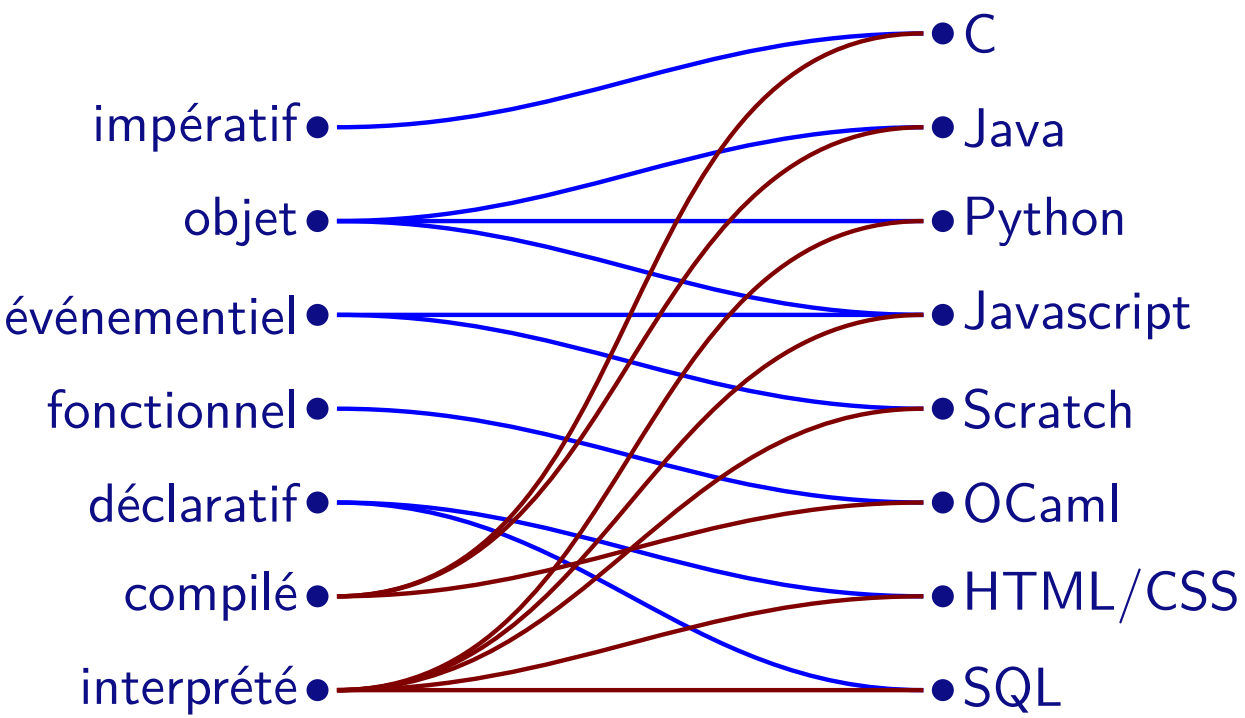
- impératif
- objet
- événementiel
- fonctionnel
- déclaratif

- C
- Java
- Python
- Javascript
- Scratch
- OCaml
- HTML/CSS
- SQL

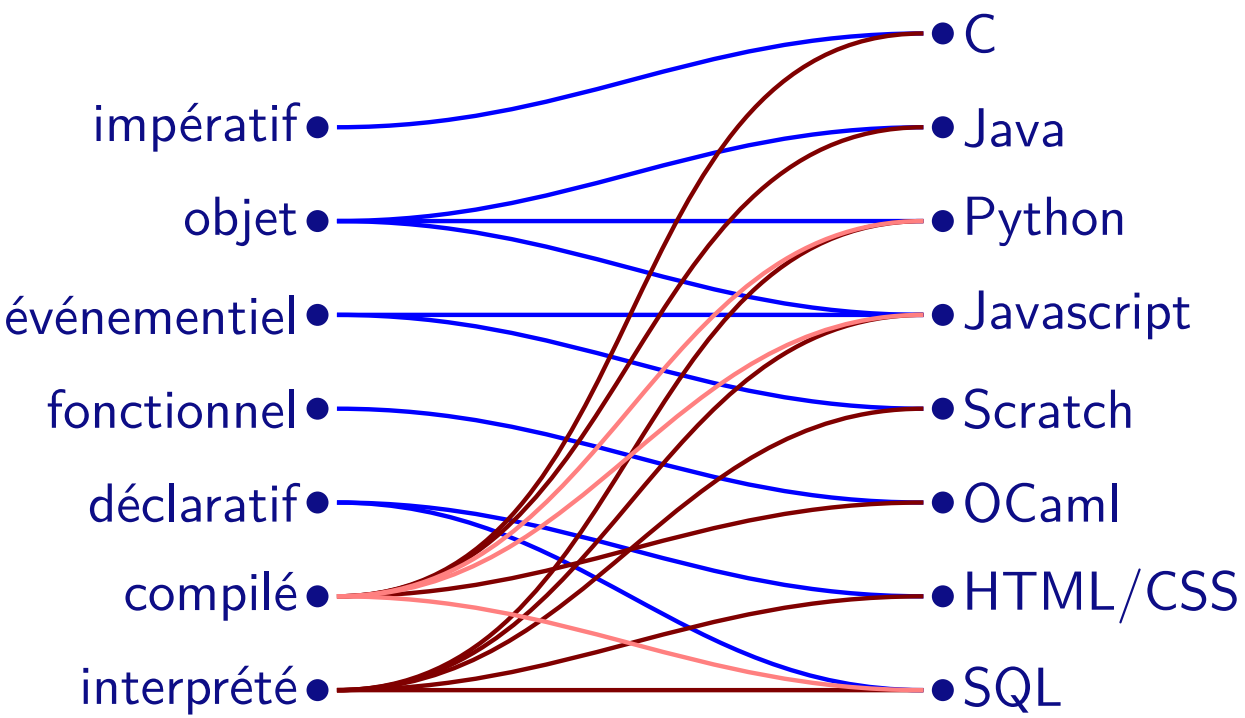
Différents paramètres



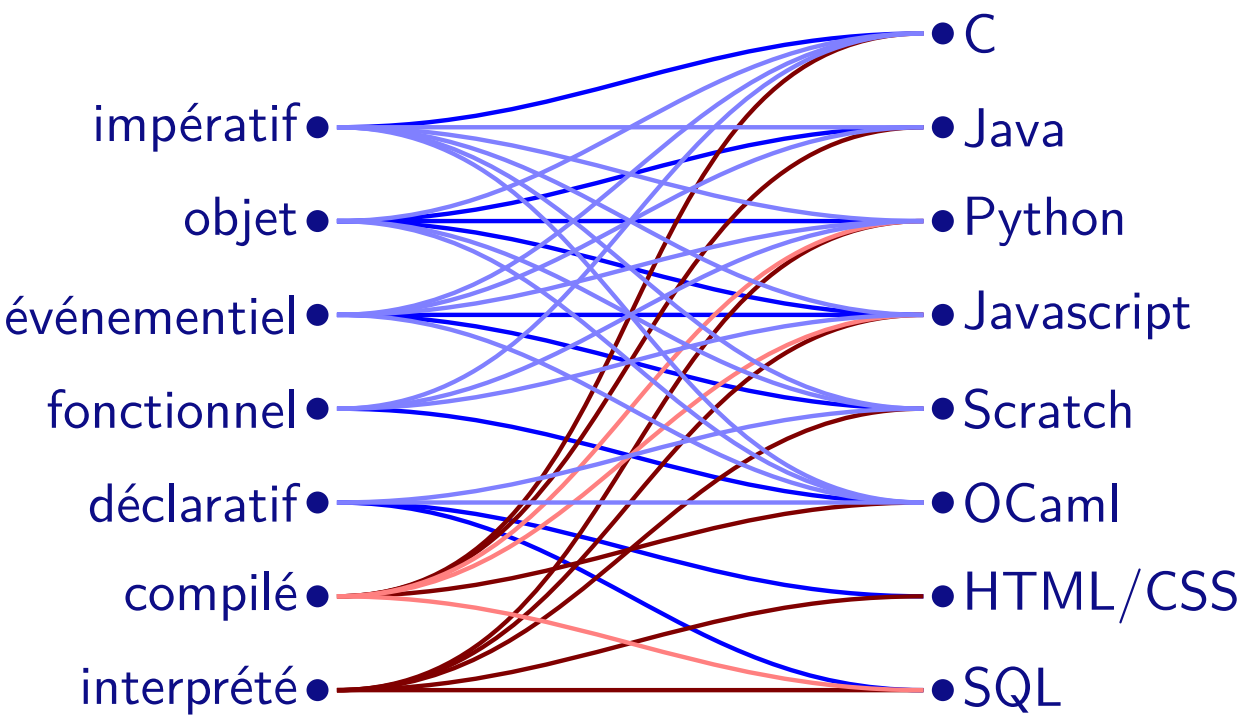
Différents paramètres



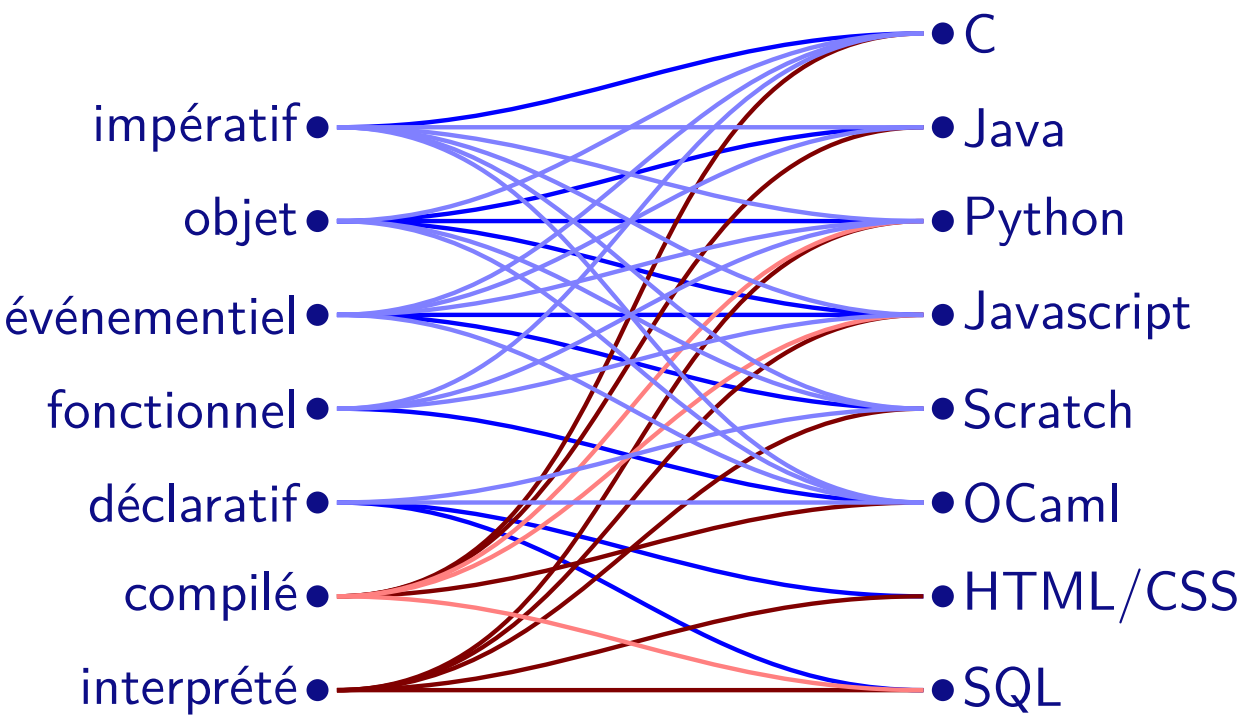
Différents paramètres



Différents paramètres



Différents paramètres



Il faut structurer un peu ...

Graphique / textuel

textuel : la plupart des langages de programmation

par blocs : Scratch, Blockly, Squeak et autres
c'est la même chose que du texte mais on clique plutôt que de taper

graphique : organigrammes, circuits, etc.
on n'en parlera pas ici...

Déclaratif / impératif / objet

Cela dépend de ce qu'on considère comme l'objet fondamental dont on parle :

machine : on donne des ordres pour effectuer une tâche
impératif, procédural

donnée : on décrit la structure des données et les traitements à faire dessus
déclaratif, fonctionnel

agent : on décrit quels sont les composants et comment ils interagissent
orienté objet, événementiel

Un calcul numérique : la factorielle

Définition mathématique « naturelle » :

$$n! = 1 \times 2 \times 3 \times \cdots \times (n-1) \times n$$

Un calcul numérique : la factorielle

Définition mathématique « naturelle » :

$$n! = 1 \times 2 \times 3 \times \cdots \times (n-1) \times n$$

Définition mathématique par récurrence :

$$n! = \begin{cases} 1 & \text{si } n \leq 1 \\ n \times (n-1)! & \text{sinon} \end{cases}$$

Un calcul numérique : la factorielle

Définition mathématique « naturelle » :

$$n! = 1 \times 2 \times 3 \times \cdots \times (n-1) \times n$$

Définition mathématique par récurrence :

$$n! = \begin{cases} 1 & \text{si } n \leq 1 \\ n \times (n-1)! & \text{sinon} \end{cases}$$

Transcription directe : déclaratif, fonctionnel

```
def fac(n):
    if n <= 1:
        return 1
    else:
        return n * fac(n - 1)
```

Un calcul numérique : la factorielle

Définition mathématique « naturelle » :

$$n! = 1 \times 2 \times 3 \times \cdots \times (n-1) \times n$$

Définition mathématique par récurrence :

$$n! = \begin{cases} 1 & \text{si } n \leq 1 \\ n \times (n-1)! & \text{sinon} \end{cases}$$

Transcription de la façon de calculer : impératif, procédural

```
def fac(n):
    f = 1
    for i in range(1, n+1):
        f = f * i
    return f
```

Résoudre un Sudoku

Une méthode très bête :

> *Prendre chaque case l'une après l'autre.*
> *Essayer chaque chiffre l'un après l'autre.*
> *Si un chiffre convient, l'écrire et passer à la case suivante, sinon revenir en arrière.*

Résoudre un Sudoku – Objet

```
class Sudoku:
    ...
    def solve(self):
        pos = self.empty_square()
        if pos is None:
            return True
        for c in range(1, 10):
            if not self.accept(pos, c):
                continue
            self.set(pos, c)
            if self.solve():
                return True
            self.clear(pos)
        return False
```

Résoudre un Sudoku – Récursif, fonctionnel

```
def solve(table):
    pos = empty_square(table)
    if pos is None:
        return table
    for c in range(1, 10):
        if not valid(table, pos, c):
            continue
        result = solve(set(table, pos, c))
        if result is not None:
            return result
    return None
```

Séquentiel / évènementiel / parallèle

Cela dépend de la façon dont s'organisent les tâches à effectuer :

séquentiel : un début, une fin, une suite d'instructions
on traite un problème pour obtenir une réponse

événementiel : le programme réagit à des événements
on interagit avec son environnement

parallèle : on décompose le programme en parties qui agissent indépendamment

Casse-brique

Description intuitive :

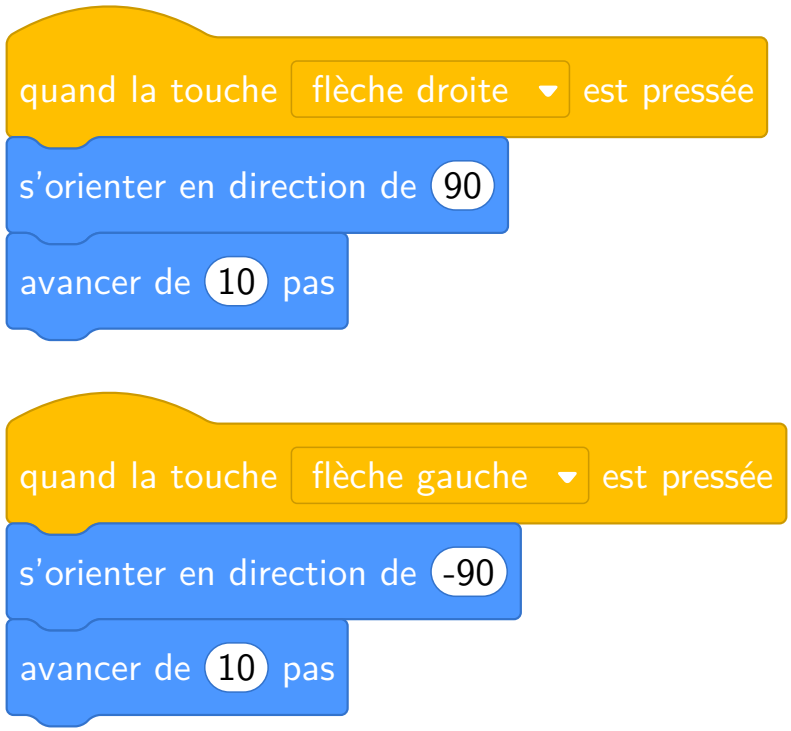
- Quand la balle touche la raquette, rebondir
- Quand la balle touche le bord, perdre une balle et reprendre
- Quand la touche *flèche droite* est pressée, déplacer la raquette vers la droite
- ...

Mise en œuvre la plus directe :

- parallèle, un composant par objet
- événementiel, pour chaque objet

Casse-brique

En Scratch, événementiel :



Casse-brique

En Python, en style objet :

```
class Raquette:
    ...
    def on_key_pressed (self, key):
        if key == RIGHT:
            self.move(10, 0)
        elif key == LEFT:
            self.move(-10, 0)
        else:
            ...
    ...
```

Casse-brique

En style impératif :

```
while True:
    key = get_next_key()
    if key == RIGHT:
        move_paddle(10, 0)
    elif key == LEFT:
        move_paddle(-10, 0)
    else:
        ...
    ...
```

Et pour enseigner ?

- Quels paradigmes enseigner pour eux-mêmes ?
- Quels paradigmes pour quelles notions ?
- Quelles progressions et quelles activités clés ?