Création de Méthodes en Ruby :

Nous allons apprendre à organiser en Ruby grâce aux Méthodes. On découper notre programme en plusieurs morceaux, chaque morceau sera ce qu'on appelle en Ruby une Méthode, c'est plus au moins l'équivalant d'une Fonctions dans d'autres langages de programmation mais qui fait appel à une classe.

Une méthode permet d'exécuter un **bout de code** (des actions) et renvoie un résultat. En ruby elle commence toujours avec ***def*** et se ferme avec ***end*** .

test-rub.rb

```ruby
def hello_world
    puts "hello world"
end
```

Ici, quand la Méthode "hello_world " va être appelé, la phrase "hello world " sera affiché.

Le nom de la méthode est juste après le mot-clé ***def***, une méthode peut lire n'importe quel code Ruby et l'exécuter (même un calcul simple comme 2 + 5 par exemple).

Comme tous les langages de programmation, une méthode peut recevoir en entrée des paramètres. Ce sont des variables qui récupérent une valeur donnée lorsque la méthode est appelée.

Exemple :

La variable result appelle la méthode multiply et passe paramètres 2 variable a et b. La méthode renvoi avec un return le résultat qui est ensuite lu avec ***puts***.

```ruby
def multiply(a, b)
    return a * b
end
result = multiply(3, 4)
puts result
# Affiche "12"
```

en

Bon tout ça, c'est bien gentil, mais quels sont les avantages des méthodes en Ruby par rapport à d'autres langages de programmation si c'est comme des fonctions ?

J'y viens, maintenant que j'ai bien rappelé ce qu'étais une méthode, on va pouvoir voir des cas un peu plus complexes, sans aller trop loin pour que ça reste dans le cadre de notre tutoriel.

Pour faire un petit rappel, le langage Ruby est un langage basé quasi entièrement sur des objets. C'est pour cela que vais aborder des méthodes qui permettent de mieux gérer ces objets.

Les blocks

Les blocks sont une fonctionnalité très populaire en Ruby, c'est un morceau de code qui est appelé dans une méthode à laquelle elle est fournie. Il se définit entre accolade { } ou ***do*** et ***end*** s'il est multi-lignes.
Toutes les fonctionnalités peuvent prendre un block et un seul comme argument, il sera interprété dans la méthode si elle fait appel au mot-clé ***yield*** pour évaluer le block.

Exemple :

```ruby
def hello_world
    hello = 'Hello'
    world = 'World!'

    yield(hello, world)
end
hello_world { |hello, world| puts "#{hello} #{world}" }
# Affiche "Hello World!"
```

Dans cet exemple, lors de l'appel de la méthode, un block est passé en paramètres avec deux paramètres à récupérer puis exécuter le code ***puts***, le block avec ***yield*** va récupérer les valeurs avant d'afficher le message lors de l'appel à la méthode.

D'autres Exemples Utiles avec les paramètres

Il est possible de définir des paramètres optionnels dans les méthodes en Ruby, comme ceci :

```ruby
def greet(name, options = {})
  greeting = options[:greeting] || "Bonjour"
  "#{greeting}, #{name}!"
end

puts greet("Marc", greeting: "Salut")
# Affiche "Salut, Marc!"
puts greet("Anne")
# Affiche "Bonjour, Anne!"
```

Le paramètre "greeting" est en options donc n'est pas obligatoire dans l'appel à la méthode, s'il n'y en a pas, il récupérera la valeur définie par défaut (ici "Bonjour") dans le message, si un paramètre est entré alors c'est ce dernier qui prendra la priorité.

Également, il est possible de regrouper plusieurs paramètres en une seul variable avec le préfixe *. Comme dans l'exemple ci-dessous.

```ruby
def addition(*nombres)
    nombres.sum
end
resultat = addition(10, 54, 23, 8)
puts resultat
# Affiche "95"
```

Toutes les valeurs rentrent dans la variable résultat lors de l'appel de la méthode addition vont être regroupé dans un tableau qui va être stocké dans une variable et effectuer l'opération.

Voilà ce qui conclut ce rapide tutoriel sur les Méthodes en Ruby, J'espère que ça vous à plus et appris 2 - 3 trucs utiles (ou pas vraiment, car y plus que 3 personnes qui code en Ruby vanilla, mais ils sont très heureux.).