ICC > func > func.py > ...

```python
def hello_world() -> None:
    print("Hello, world!")
```

func.java

ICC > func > func.java

```java
void helloWorld() {
    System.out.println("Hello, world!");
}
```

func.cpp

ICC > func > func.cpp

```cpp
#include <iostream>

void hello_world() {
    std::cout << "Hello, world!";
}
```

func.lisp

ICC > func > func.lisp

```lisp
(defun hello_world () (print "Hello, world!"))
```

Information, Calcul et Communication

Partie Programmation

Cours 3: Structures de contrôle (fin) – Fonctions

06.10.2023

Patrick Wang

Au sommaire cette semaine

1. Structures de contrôle – suite et fin
2. Définitions et appels de fonctions

Au sommaire cette semaine

1. Structures de contrôle – suite et fin
2. Définitions et appels de fonctions

1. Structures de contrôle – suite et fin

Récapitulatif et petit extra

- Algèbre booléenne et instructions conditionnelles
- Que va produire le programme suivant ?
- Question sous-jacente : Est-ce que les messages seront affichés en double lorsque $i < 20$?

```
for i in range(30):
    if i < 10:
        print(f"i < 10, {i}")
    elif i < 20:
        print(f"i < 20, {i}")
    else:
        print("else")
```

1. Structures de contrôle – suite et fin

Est-ce que les messages seront affichés en double lorsque $i < 20$?

- La réponse est non !
- Exemple de lecture du programme :

```
SI la première condition est vraie ALORS:
    Faire les instructions
    Quitter toute l'instruction conditionnelle
SINON SI la condition précédente est fausse
      ET la seconde condition est vraie ALORS :
    Faire les instructions
    Quitter toute l'instruction conditionnelle
SINON (SI toutes les conditions précédentes sont fausse) ALORS:
    Faire les instructions
    Quitter toute l'instruction conditionnelle
```

```python
for i in range(30):
    if i < 10:
        print(f"i < 10, {i}")
    elif i < 20:
        print(f"i < 20, {i}")
    else:
        print("else")
```

1. Structures de contrôle – suite et fin

La boucle `for`

- Deux utilisations de la boucle `for` vue en cours puis en exercices :
  - Répéter un bloc d'instructions (exemple du cours)
  - Parcourir les éléments d'un «objet itérable» (s02e05 : Manipulation de chaînes de caractères)
- En fait, ces deux utilisations sont équivalentes parce que l'on sait à l'avance combien de répétitions seront faites.
- Une éventuelle différence serait la façon dont on utilise la variable de boucle dans le corps de la boucle.

1. Structures de contrôle – suite et fin

La boucle `for`

Variable de boucle qui n'est jamais utilisée dans le corps de la boucle

```
for i in range(14):
   if month in (1, 3, 5, 7, 8, 10, 12):
        nb_days_in_month = 31
    elif month in (4, 6, 9, 11):
        nb_days_in_month = 30
    else:
        nb_days_in_month = 28
    if day + 7 > nb_days_in_month:
        day = (day + 7) - nb_days_in_month
        month += 1
    else:
        day += 7
```

Variable de boucle qui est réutilisée dans le corps de la boucle

```
line: str = input("Un message: ")

uppercase: int = 0
lowercase: int = 0

for char in line:
    if char.islower():
        lowercase += 1
    elif char.isupper():
        uppercase += 1
print(f"Majuscules: {uppercase}")
print(f"Minuscules: {lowercase}")
```

1. Structures de contrôle – suite et fin

La boucle `for` – élément ou indice ?

```python
my_str: str = "Un bout de texte"
for char_index in range(len(my_str)):
    # On récupère le caractère à l'indice char_index
    # Puisque char_index va de 0 à len(my_str) - 1,
    # on parcourt ainsi toute la chaîne.
    print(my_str[char_index])

for char in my_str:
    # On récupère directement chaque caractère les uns après les autres.
    # On a perdu l'information de l'indice de cette façon.
    print(char)

for index, char in enumerate(my_str):
    print(index, char)
```

1. Structures de contrôle – suite et fin

La boucle `while`

- TANT QUE <expression booléenne> RÉPÉTER :
- La boucle while va donc répéter les instructions situées dans le corps de la boucle tant qu'une condition est vraie

Condition de répétition

```
course_number: int = 1
while course_number <= 14:
    print(f"{day}.{month}.{year}")
    # Calcul de la date du prochain cours ICC...
    course_number += 1
```

Il faut manuellement incrémenter la variable de boucle

1. Structures de contrôle – suite et fin

Boule `for` ou `while` ?

- Il existe deux types de boucles. Quelles sont les différences ? Comment savoir quand utiliser plutôt l'une que l'autre ?
- Boucle `for`:
  - Utile pour parcourir un objet itérable (`str`, `range()`, …)
  - On sait d'avance combien d'itérations seront réalisées
- Boucle `while`:
  - Utile si une condition doit être vérifiée pour répéter les instructions
  - Peut faire le même travail qu'une boucle `for`
  - Peut également construire des boucles infinies

1. Structures de contrôle – suite et fin

Bilan

- 11 mots-clés sur 35 ! Et en plus, on ne va pas tous les rencontrer cette année !

2.3.1. Keywords

The following identifiers are used as reserved words, or *keywords* of the language, and cannot be used as ordinary identifiers. They must be spelled exactly as written here:

```
False ✓    await      else ✓     import     pass
None       break      except     in ✓       raise
True ✓     class      finally    is         return
and ✓      continue   for ✓      lambda     try
as         def        from       nonlocal   while ✓
assert     del        global     not ✓      with
async      elif ✓     if ✓       or ✓       yield
```

Au sommaire cette semaine

1. Structures de contrôle – suite et fin
2. Définitions et appels de fonctions

2. Définitions et appels de fonctions

« Des mots créés pour nous »

- On a déjà utilisé pas mal de fonctions depuis le début du semestre :
  - `print()`
  - `type()`
  - `len()`
  - `input()`
  - `range()`
  - `enumerate()`
  - `int()`, `float()`, `str()`
- Les fonctions de la « bibliothèque standard » : https://docs.python.org/3/library/functions.html (et il en existe d'autres dans des « modules »)

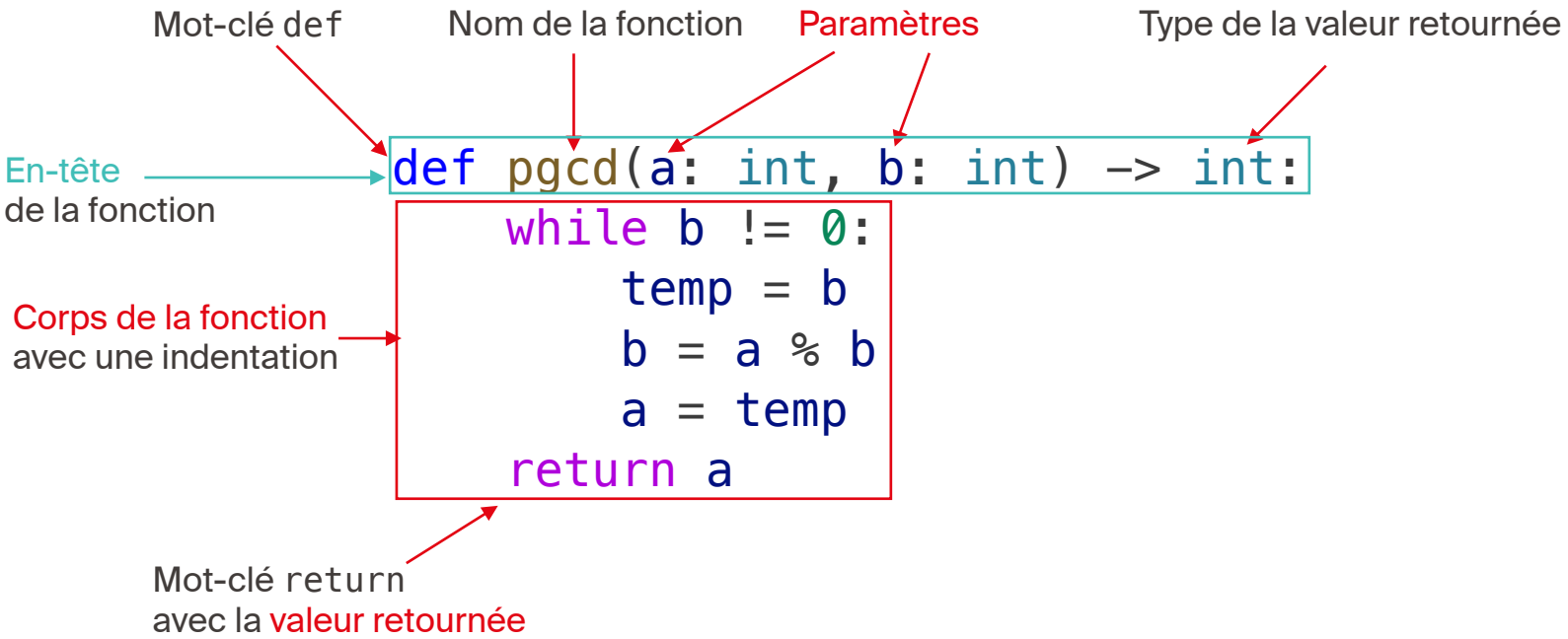
2. Définitions et appels de fonctions

C'est quoi une fonction ?

- Une fonction est une séquence d'instructions à laquelle on a donné un nom.
- Définir une fonction, c'est dire à Python qu'une séquence d'instructions particulière possède désormais un nom.
- Appeler une fonction, c'est demander à ce que toute la séquence d'instructions soit exécutée en lieu et place de l'appel de la fonction.

2. Définitions et appels de fonctions

Comment définir une fonction ?




```
def pgcd(a: int, b: int) -> int:
    while b != 0:
        temp = b
        b = a % b
        a = temp
    return a
```

2. Définitions et appels de fonctions

Comment définir une fonction ?

- Les éléments indispensables dans la définition d'une fonction :
  - Mot-clé `def`
  - Nom de la fonction
  - Parenthèses (éventuellement vides)
  - “Deux-points”
  - Corps de la fonction (éventuellement sans `return`)

```
def useless_function():
    print("I'm not useless!")
```

2. Définitions et appels de fonctions

Comment définir une fonction ?

- Les éléments indispensables dans la définition d'une fonction :
  - Mot-clé `def`
  - Nom de la fonction
  - Parenthèses (éventuellement vides)
  - “Deux-points”
  - Corps de la fonction (éventuellement sans `return`)

```
def useless_function() -> None:
    print("I'm not useless!")
```

2. Définitions et appels de fonctions

Comment appeler une fonction ?

```python
def pgcd(a: int, b: int) -> int:
    while b != 0:
        temp = b
        b = a % b
        a = temp
    return a

def useless_function() -> None:
    print("I'm not useless!")

useless_function()
resultat: int = pgcd(525, 81)
print(resultat)
```

2. Définitions et appels de fonctions

Comment appeler une fonction ?

Définitions de fonctions
(idéalement en début de fichier)

```python
def pgcd(a: int, b: int) -> int:
    while b != 0:
        temp = b
        b = a % b
        a = temp
    return a

def useless_function() -> None:
    print("I'm not useless!")

useless_function()
resultat: int = pgcd(525, 81)
print(resultat)
```

2. Définitions et appels de fonctions

Comment appeler une fonction ?

Définitions de fonctions
(idéalement en début de fichier)

Appel d'une fonction
sans valeur retournée

```
def pgcd(a: int, b: int) -> int:
    while b != 0:
        temp = b
        b = a % b
        a = temp
    return a


def useless_function() -> None:
    print("I'm not useless!")

useless_function()
resultat: int = pgcd(525, 81)
print(resultat)
```

2. Définitions et appels de fonctions

Comment appeler une fonction ?

Définitions de fonctions
(idéalement en début de fichier)

Appel d'une fonction
sans valeur retournée

Appel d'une fonction avec
valeur retournée (et assignée
à une variable)

```python
def pgcd(a: int, b: int) -> int:
    while b != 0:
        temp = b
        b = a % b
        a = temp
    return a

def useless_function() -> None:
    print("I'm not useless!")

useless_function()
resultat: int = pgcd(525, 81)
print(resultat)
```

2. Définitions et appels de fonctions

Explications sur le mot-clé `return`

- Le mot-clé `return`, lorsque présent, est suivi de la valeur calculée et retournée par la fonction
- Lorsqu'une fonction est appelée, l'exécution d'une instruction contenant le mot-clé `return` va terminer l'exécution de cette fonction.

```python
def useless_instructions_after_return() -> int:
    return 1
    print("A useless instruction")
    print("Another useless instruction")
    return 4
```

2. Définitions et appels de fonctions

Point sur le vocabulaire

- 14 mots-clés sur 35 ! On a vu les mots-clés les plus importants du semestre.
- Il y en aura tout de même d'autres que l'on va croiser.

2.3.1. Keywords

The following identifiers are used as reserved words, or *keywords* of the language, and cannot be used as ordinary identifiers. They must be spelled exactly as written here:

```
False ✓   await      else ✓    import     pass
None ✓    break      except    in ✓       raise
True ✓    class      finally   is         return ✓
and ✓     continue   for ✓     lambda     try
as        def ✓      from      nonlocal   while ✓
assert    del        global    not ✓      with
async     elif ✓     if ✓      or ✓       yield
```

Bilan

Ingrédients de base des algorithmes

- Cours 1:
  - Séquences d'instructions
  - Variables
  - Quelques opérateurs et fonctions utiles
- Cours 2:
  - Algèbre booléenne
  - Instructions conditionnelles (`if – elif – else`)
  - Instructions itératives (`for`)
- Cours 3:
  - Structures de contrôle (suite et fin)
  - Définition et appel de fonctions

Séances d'exercices

- CM 1 112 avait un peu trop de monde
- Il restait environ 8 places en CM 1 110, et peu en CM1 103.