Partie I

Introduction à la programmation

Informatique 1
Introduction à la programmation

Mission 3 : INTRODUCTION

Fonctions, spécifications, modules, graphiques tortue

Kim Mens Siegfried Nijssen Charles Pecheur

Appel de fonction

Que se passe-t-il vraiment quand on exécute

```
n = 42
print("n vaut", n)
```

Appelle la **fonction** `print`
avec les **arguments** `"n vaut"` et `42`

Définition de fonction

```
def hello_world():
    """Affiche le message 'Hello, world!'
    """
    print("Hello, world!")
```

en-tête

documentation

corps

```
>>> hello_world()
Hello, world!
```

appel

Fonction avec paramètres

paramètre : une **variable**

```
def hello(name):
    """Affiche le message 'Hello, XXX'
       où XXX est donné par `name`
    """
    print("Hello,", name)
```

argument : une **expression**

```
>>> hello("Charles")
Hello, Charles
```

Fonction avec résultat

fonction fructueuse
Ch. 6 – Fruitful functions

```
def hello_message(name):
    """Retourne le string 'Hello, XXX'
       où XXX est donné par `name`
    """
    message = "Hello, " + name
    return message
```

retourne `message`
+ sort de la fonction

```
>>> msg = hello_message("Charles")
>>> print(msg)
Hello, Charles
```

appel = expression

return

```
def test_premier(n):
    for k in range(2, n):
        if n%k == 0:
            print(n, " n'est pas premier")
            return
    print(n, " est premier")
```

Sort de la fonction

Définition de fonction

```
def hello_message(name):
    """Retourne le string 'Hello, XXX'
       où XXX est donné par `name`
    """
    message = "Hello, " + name
    return message
```

```
>>> hello_message("Charles")
Hello, Charles!
```

Définition de fonction

(1) En-tête

```
def hello_message(name):
    """Retourne le string 'Hello, XXX'
       où XXX est donné par `name`
    """
    message = "Hello, " + name
    return message
```

```
>>> hello_message("Charles")
Hello, Charles!
```

Définition de fonction

(1) En-tête
(2) Spécifications

```
def hello_message(name):
    """Retourne le string 'Hello, XXX'
       où XXX est donné par `name`
    """
    message = "Hello, " + name
    return message
```

```
>>> hello_message("Charles")
Hello, Charles!
```

Définition de fonction

(1) En-tête

(2) Spécifications

(3) Corps

```
def hello_message(name):
    """Retourne le string 'Hello, XXX'
       où XXX est donné par `name`
    """
    message = "Hello, " + name
    return message
```

```
>>> hello_message("Charles")
Hello, Charles!
```

Définition de fonction

(1) En-tête

(2) Spécifications

```
def hello_message(name):
    """Retourne le string 'Hello, XXX'
       où XXX est donné par `name`
    """
    message = "Hello, " + name
    return message
```

(3) Corps

(4) Return

```
>>> hello_message("Charles")
Hello, Charles!
```

Définition de fonction

(1) En-tête

(2) Spécifications

```
def hello_message(name):
    """Retourne le string 'Hello, XXX'
       où XXX est donné par `name`
    """
    message = "Hello, " + name
    return message
```

(3) Corps

(4) Return

```
>>> hello_message("Charles")
Hello, Charles!
```

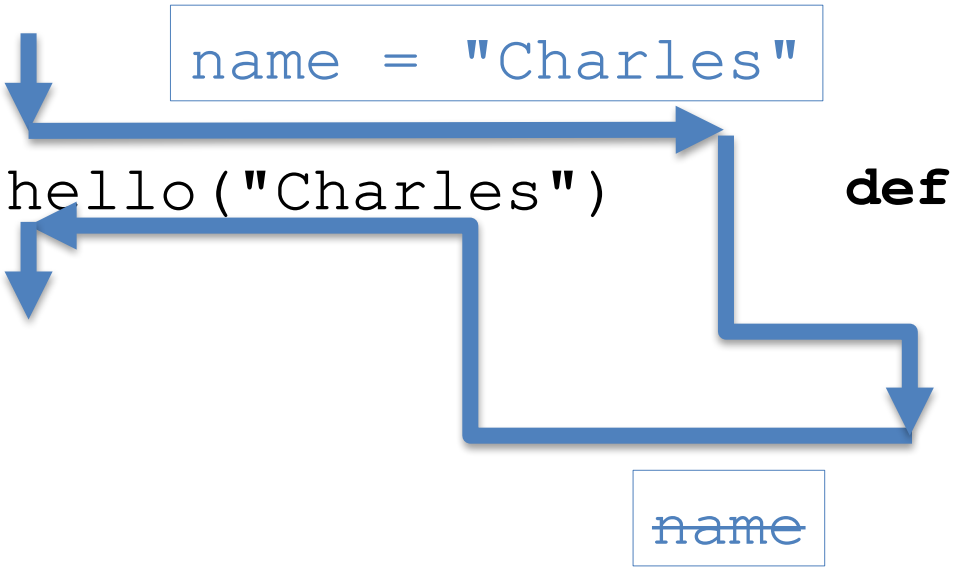
appel

Appel de fonction

Que se passe-t-il vraiment quand on exécute

```
hello("Charles")
```

Dans Thonny:




```
Hello, Charles
```

Appel d'une fonction

```
def fonction(paramètres):
    """documentation"""
    corps

fonction(arguments)
```

- Les **paramètres** prennent la **valeur des arguments**
- Le **corps** de la fonction est exécuté
- L'exécution continue après l'appel

Docstring

```
def hello(name):
    """Affiche le message 'Hello, XXX'
       où XXX est donné par `name`
    """
    print("Hello,", name)
```

String en tête de définition, documente la fonction

```
>>> help(hello)
Help on function hello in module __main__:

hello(name)
    Affiche le message 'Hello, XXX'
    où XXX est donné par `name`
```

help(print)

>>> help(print)

Help on built-in function print in module builtins:

print(...)

print(value, ..., **sep=' '**, **end='\n'**, file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default.

Optional keyword arguments:

file: a file-like object (stream); defaults to the current sys.stdout.

sep: string inserted between values, default a space.

end: string appended after the last value, default a newline.

flush: whether to forcibly flush the stream.

print()

```
for k in range(10):
    print(k, end=".")
print()
```

```
0.1.2.3.4.5.6.7.8.9.
```

```
print(720, 22, 11, sep="-")
```

```
720-22-11
```

Pas dans le syllabus

Spécification d'une fonction

Spécifier une fonction =

décrire rigoureusement l'effet d'une fonction

Pré-conditions = Conditions initiales

Les conditions sous lesquelles la fonction est applicable

Post-conditions = Résultat

Ce que fera ou retournera la fonction exécutée

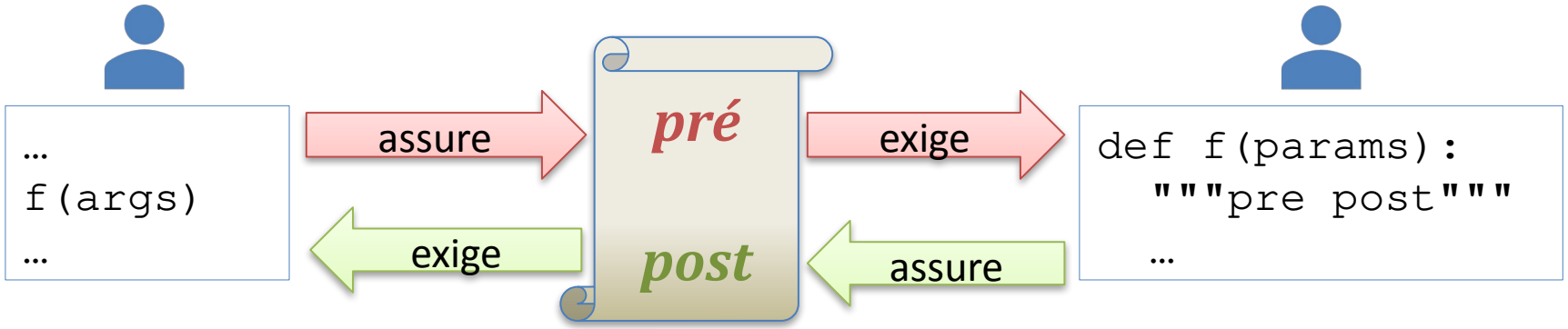
```
def hello(name):
    """pre: `name` est un string
       post: Affiche le message 'Hello, XXX'
       où XXX est donné par `name`
    """
    print("Hello,", name)
```

Contrat d'une fonction

La spécification est un **contrat**
entre la **fonction** et ses **utilisateurs**

Contrat de la fonction

*Si **vos données** satisfont mes **pré-conditions**,*
*Alors **mes résultats** satisferont mes **post-conditions**.*



Spécifications : exemples

Racine carrée d'un nombre

```
def racine(n):
    """pre: n>=0
       post: retourne la racine carrée de n
    """
```

Carré d'un nombre

```
def carre(n):
    """pre: -
       post: retourne le carré de n
    """
```

Nom d'un mois

```
def mois(n):
    """pre: n entre 1 et 12
       post: retourne le nom du n-ième mois
    """
```

Variables locales

Variables locales d'une fonction

- Les **paramètres** de la fonction
- Les **variables affectées** dans la fonction

- Il ne sont **pas accessibles en-dehors** de la fonction
- Ils **cessent d'exister** quand la fonction se termine

```
k = 100
r = racine(150)
print(r)
print(k)
print(n)
```

12

100

ERREUR

n et **k** sont **indéfinis**

n et **k** sont **définis** et **utilisables**

```
def racine(n):
    """..."""
    k = 0
    while k*k <= n:
        k += 1
    return k-1
```

Modules

Un **module** est un fichier Python contenant
un ensemble de définitions et d'instructions

La **librairie standard Python** offre un **vaste choix de modules**

texte, calcul, fichiers, réseau, web, graphique, système, …

```
>>> help("modules")
```

https://docs.python.org/3/library

On peut **installer des modules** supplémentaires

par exemple : **`numpy`** (calculs), **`matplotlib`** (graphiques)

installer : **`pip`**, Thonny package manager

Pour utiliser le module **`turtle`** dans un programme :

```
import turtle
```

Module `math`

```
>>> import math
>>> print(math.pi)
                    3.141592653589793
>>> print(math.cos(math.pi/3))
                    0.5000000000000001
>>> print(math.cos(math.radians(60)))
                    0.5000000000000001
>>> print(math.sqrt(2.0))
                    1.4142135623730951
>>> print(math.log(math.e))
                    1.0
```

Accéder au module **math**

help(math)

>>> help(math)

Help on module math:

NAME
  math
DESCRIPTION
  This module is always available. It provides access to the
  mathematical functions defined by the C standard.
FUNCTIONS
  acos(...)
    acos(x)
    Return the arc cosine (measured in radians) of x.
acosh(...)
    acosh(x)
    Return the inverse hyperbolic cosine of x.
[……]
DATA
  e = 2.718281828459045
  inf = inf
  nan = nan
  pi = 3.141592653589793
  tau = 6.283185307179586

>>> help(math.sqrt)

Help on built-in function sqrt in module math:

sqrt(...)
  sqrt(x)
  Return the square root of x.

>>> help(math.log)

Help on built-in function log in module math:

log(...)
  log(x[, base])
  Return the logarithm of x to the given base.
  If the base not specified, returns the natural logarithm (base e) of x.

Graphiques tortue

```
import turtle
tortue = turtle.Turtle()
tortue.color("blue")
for i in range(4):
    tortue.forward(50)
    tortue.right(90)
```

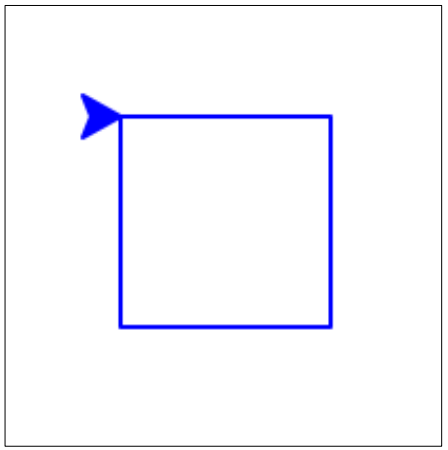
Module des graphiques tortue

Crée une nouvelle tortue (**objet**)

appelle une fonctionalité de la tortue (**méthode**)

idem.

idem.



Objets

Une tortue est un **objet** (de type `turtle.Turtle`)

Une **instance** de `turtle.Turtle`

Un **objet** est une structure qui associe

- des **données** (**attributs**)
  - position, orientation de la tortue
  - position haut/bas, couleur, largeur de la plume
  - …
- des **fonctionnalités** (**méthodes**)
  - avancer, reculer, tourner à gauche/droite
  - lever/baisser la plume, changer de couleur
  - remplir un contour
  - …

Méthodes

```
tortue.forward(50)
```

Exécute (invoque) la **méthode** forward de **l'objet** tortue

Méthode = **fonction liée à un objet**

L'objet est un paramètre de la fonction :

```
def forward(self, distance): …
```

Méthodes de `Turtle`

```
tortue.forward(10), tortue.backward(10)
tortue.left(45), tortue.right(45)
tortue.circle(20)
tortue.color("blue")
tortue.pendown(), tortue.penup()
tortue.begin_fill(), tortue.end_fill()
```

… et beaucoup d'autres

Voir `help(turtle)`

Mission 3

Objectifs

- fonctions
- spécifications
- décomposition en sous-problèmes

Problème

- Dessiner un drapeau