UNIVERSITÉ DE LORRAINE

Institut des sciences du Digital Management & Cognition
COMPOSANTE DE L'UNIVERSITÉ DE LORRAINE

Programming

4- Functions

Those slides will be available on Arche

By Gaël Guibon, inspired by Mathieu d'Aquin's course

Functions

What is a function?

A washing machine is a function

Instructions:
- Pour water
- Wash
- Dry

Arguments:
- Clothes
- Soap

Returns (output):
- Clean clothes

What is a function?

A rice cooker is a function

Instructions:

- Boil elements while minutes < duration

Arguments:

- Rice
- (optional) duration (default: 20 minutes)
- (optional) salt
- (optional) chicken

Returns:

- Cooked elements

What is a function?

Drinking a bubble tea is a function

Instructions:

- Grab bubble tea
- Drink until there is no 🧋 anymore

Arguments:

- Bubble tea size (ml)

Returns:

- Empty container

What is a function?

Drinking a bubble tea is a function

```python
def drink_bubbletea(size=500):
    """drinks a bubble tea sip by sip"""
    sip_ml = 10
    size_ml = size
    while size_ml > 0:
        size_ml -= sip_ml
    return size
```

Functions

You have been using many functions already

- `print()` and `input()` are “builtin” functions
- `lower()` and `upper()` are functions that apply on strings
- `abs()`, `pow()` are functions from the `math` library

They are way to make code “callable” so that code can be **reused**. It also allows to structure the overall script/programme through **decomposition** and **abstraction**.

Decomposition and Abstraction Using Functions

In your game, you print the player status (healthpoints, etc.) often.
Use a function:

```
def show_status(player_stats):
    print("="*5, player_stats['name'], "="*5)
    for k, v in player_stats.items():
        print(f'{k}: {v}')
    print("="*20)
```

Decomposition and Abstraction Using Functions

In your game, you print the player status (healthpoints, etc.) often.
Use a function:

Decomposition: This allows not to repeat the same logic over and over.


```python
def show_status(player_stats):
    print("="*5, player_stats['name'], "="*5)
    for k, v in player_stats.items():
        print(f'{k}: {v}')
    print("="*20)

show_status(cloud_stats)
show_status(tifa_stats)
```

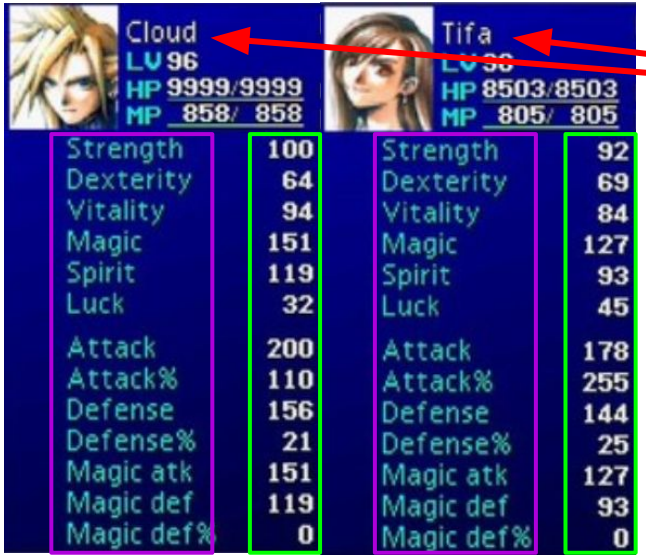
Decomposition and Abstraction Using Functions

In your game, you print the player status (healthpoints, etc.) often.

Use a function:

Decomposition: This allows not to repeat the same logic over and over.

key : value




```python
def show_status(player_stats):
    print("="*5, player_stats['name'], "="*5)
    for k, v in player_stats.items():
        print(f'{k}: {v}')
    print("="*20)

show_status(cloud_stats)
show_status(tifa_stats)
```

Decomposition and Abstraction Using Functions

In your game, you print the player status (healthpoints, etc.) often.
Use a function:

Abstraction: allows to use the function for any character (Cloud, Barret or Tifa)

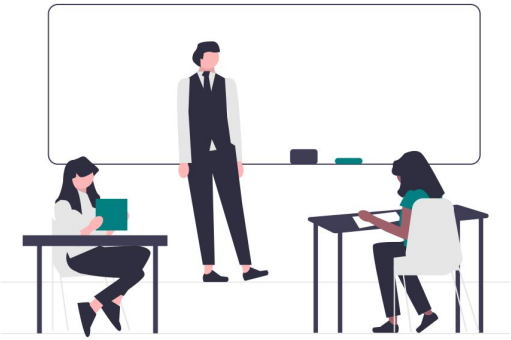

```python
def show_status(player_stats):
    print("="*5, player_stats['name'], "="*5)
    for k, v in player_stats.items():
        print(f'{k}: {v}')
    print("="*20)

show_status(cloud_stats)
show_status(tifa_stats)
```

Decomposition and Abstraction Using Functions

In your game, you print the player status (healthpoints, etc.) often.
Use a function:

Abstraction: allows to use the function for **any set of statistics** (even students)



```python
def show_status(player_stats):
    print("="*5, player_stats['name'], "="*5)
    for k, v in player_stats.items():
        print(f'{k}: {v}')
    print("="*20)

show_status(marion_stats)
show_status(fantine_stats)
show_status(nicolas_stats)
```

Abstraction

Drinking a **something** is a function

Variable content

Instructions:

- Grab the **drink container**
- Drink until there is no **liquid** anymore

Arguments:

- **Drink** size (ml)

Returns:

- Empty container

Functions are subalgorithms

Functions can return values.

Input: l - a list, x - element to search
Output: Index of x in l

```
for each element e at index i in l do
   if x == e then
      show(i)
      finish
   endif
done
```

Functions are subalgorithms

Functions can return values.

```
def findInList(l,x):
 """finds the element x in the list l"""
 i = 0
 while i < len(l):
   if l[i] == x:
     return i
   i = i + 1
 return None
```

Remember the program from the first lecture?

```
# displays n candles
def candles(n):
    l1 = l2 = ""
    for i in range(0,n):
      l1 += "()"
      l2 += "||"
    print(l1)
    print(l2)

candles(42)
```

Remember the program from the first lecture?

```
# displays n candles
def candles(n):
    l1 = l2 = ""
    for i in range(0,n):
      l1 += "()"
      l2 += "||"
    print(l1)
    print(l2)

candles(42)
```

Remember the program from the first lecture?

```
from datetime import datetime
# tries to parse the DoB and returns the number of years (0 if error)
# and whether this is today
def parseDOB(dobString):
  try:
     date = datetime.strptime(dobString,  "%d/%m/%y")
  except:
     return 0,False
  now = datetime.now()
  diff = int((now - date).days) //  365
  return diff,(date.day==now.day  and date.month==now.month)

print(parseDOB("12/05/79"))
```

Remember the program from the first lecture?

```
from datetime import datetime
# tries to parse the DoB and returns the number of years (0 if error)
# and whether this is today
def parseDOB(dobString):
  try:
     date = datetime.strptime(dobString, "%d/%m/%y")
  except:
     return 0,False
  now = datetime.now()
  diff = int((now - date).days) // 365
  return diff,(date.day==now.day and date.month==now.month)

print(parseDOB("12/05/79")) >>> (42, False)
```

Remember the program from the first lecture?

```
from datetime import datetime
# tries to parse the DoB and returns the number of years (0 if error)
# and whether this is today
def parseDOB(dobString):
  try:
     date = datetime.strptime(dobString, "%d/%m/%y")
  except:
     return 0,False
  now = datetime.now()
  diff = int((now - date).days) // 365
  return diff,(date.day==now.day and date.month==now.month)

print(parseDOB("12/05/79")) >>> (42, False)
```

Functions

functions are ways to “name” code that can then be called by name.

```
def numbersto10():
 print("hey, I'm in the function.")
 for x in range(11):
   print(x)
```

---------*some time later*---------

```
numbersto10()
```

Functions

functions are ways to “name” code that can then be called by name.

```
def numbersto10():
 print("hey, I'm in the function.")
 for x in range(11):
   print(x)
```

---------*some time later*---------

```
numbersto10()
```

```
hey, I'm in the function.
0
1
2
3
4
5
6
7
8
9
10
```

Functions Parameters

functions can take parameters

```
def numbersto(n):
 print("hey, I'm in the second function.")
 for x in range(n+1):
   print(x)
```

---------*some time later*---------

```
numbersto(5)
```

Functions Parameters

functions can take parameters

```
def numbersto(n):
 print("hey, I'm in the second function.")
 for x in range(n+1):
   print(x)
```

---------*some time later*---------

```
numbersto(5)
```

```
hey, I'm in the second
function.
0
1
2
3
4
5
```

Functions Parameters

functions can take parameters

```
def numbersfromto(s,n):
 print("hey, I'm in the third function.")
 for x in range(s,n+1):
   print(x)
```

---------*some time later*---------

```
numbersfromto(3,8)
```

Functions Parameters

functions can take parameters

```
def numbersfromto(s,n):
 print("hey, I'm in the third function.")
 for x in range(s,n+1):
   print(x)
```

---------*some time later*---------

```
numbersfromto(3,8)
```

```
hey, I'm in the third
function.
3
4
5
6
7
8
```

Functions Parameters

player_stats is a **parameter**

variables or values can be given as parameters

```
def show_status(player_stats):
    print("="*5, player_stats['name'], "="*5)
    for k, v in player_stats.items():
        print(f'{k}: {v}')
    print("="*20)

show_status(cloud_stats)
show_status(tifa_stats)
```

Functions Parameters

player_stats is a **parameter**

variables or values can be given as parameters

Cloud and Tifa stats are data given to fill the player_stats parameter

key : **value**


```python
def show_status(player_stats):
    print("="*5, player_stats['name'], "="*5)
    for k, v in player_stats.items():
        print(f'{k}: {v}')
    print("="*20)

show_status(cloud_stats)
show_status(tifa_stats)
```

Functions Optional Parameters

Parameters can be **optional** and take **default values**

```
def numberstofrom(n,s=0):
 print("hey, I'm in the fourth function.")
 for x in range(s,n+1):
   print(x)
```

---------*some time later*---------

```
numberstofrom(5,3)
numberstofrom(5)
```

Functions Optional Parameters

Parameters can be **optional** and take **default values**

```
def numberstofrom(n,s=0):
 print("hey, I'm in the fourth function.")
 for x in range(s,n+1):
   print(x)
```

---------*some time later*---------

```
numberstofrom(5,3)
numberstofrom(5)
```

Functions Optional Parameters

Parameters can be **optional** and take **default values**

```
def numberstofrom(n,s=0):
 print("hey, I'm in the fourth funct
 for x in range(s,n+1):
   print(x)
```

---------*some time later*---------

```
numberstofrom(5,3)
numberstofrom(5)
```

```
hey, I'm in the fourth
function.
3
4
5
hey, I'm in the fourth
function.
0
1
2
3
4
5
```

Functions with Docstring

Functions can return values. Remember our “find in list” algorithm?

```
def findInList(l,x):
 """finds the element x in the list l"""
 i = 0
 while i < len(l):
   if l[i] == x:
     return i
   i = i + 1
 return None
```

docstring
Dedicated metadata

Functions

Functions can return values. Remember our “find in list” algorithm?

```
l = ["camille", "ilan", "pin-xun", "enzo", "ahana"]
i1 = findInList(l, "ilan")
print(i1)
i2 = findInList(l, "pin-xun")
print(i2)
i3 = findInList(l, "gael")
print(i3)
print(findInList.__doc__)
```

Functions

Functions can return values. Remember our “find in list” algorithm?

```
l = ["camille", "ilan", "pin-xun", "enzo", "ahana"]
i1 = findInList(l, "ilan")
print(i1) >>> 1
i2 = findInList(l, "pin-xun")
print(i2)
i3 = findInList(l, "gael")
print(i3)
print(findInList.__doc__)
```

Functions

Functions can return values. Remember our “find in list” algorithm?

```
l = ["camille", "ilan", "pin-xun", "enzo", "ahana"]
i1 = findInList(l, "ilan")
print(i1) >>> 1
i2 = findInList(l, "pin-xun")
print(i2) >>> 2
i3 = findInList(l, "gael")
print(i3)
print(findInList.__doc__)
```

Functions

Functions can return values. Remember our “find in list” algorithm?

```
l = ["camille", "ilan", "pin-xun", "enzo", "ahana"]
i1 = findInList(l, "ilan")
print(i1) >>> 1
i2 = findInList(l, "pin-xun")
print(i2) >>> 2
i3 = findInList(l, "gael")
print(i3) >>> None
print(findInList.__doc__)
```

Functions

Functions can return values. Remember our “find in list” algorithm?

```
l = ["camille", "ilan", "pin-xun", "enzo", "ahana"]
i1 = findInList(l, "ilan")
print(i1) >>> 1
i2 = findInList(l, "pin-xun")
print(i2) >>> 2
i3 = findInList(l, "gael")
print(i3) >>> None
print(findInList.__doc__) >>> finds the element x in the list l
```

What does this function?

```
def myfunction(n):
 if n > 0: return n
 else: return -n
---
print(myfunction(3))
print(myfunction(-4))
print(myfunction(0))
```

Functions

Absolute value of number

```
def absoluteValue(n):
 if n > 0: return n
 else: return -n
---
print(absoluteValue(3))
print(absoluteValue(-4))
print(absoluteValue(0))
```

Functions

Absolute value of number

```
def absoluteValue(n):
 if n > 0: return n
 else: return -n
---
print(absoluteValue(3))   >>> 3
print(absoluteValue(-4))
print(absoluteValue(0))
```

Functions

Absolute value of number

```
def absoluteValue(n):
 if n > 0: return n
 else: return -n
---
print(absoluteValue(3))  >>> 3
print(absoluteValue(-4)) >>> 4
print(absoluteValue(0))
```

Functions

Absolute value of number

```
def absoluteValue(n):
 if n > 0: return n
 else: return -n
---
print(absoluteValue(3))  >>> 3
print(absoluteValue(-4)) >>> 4
print(absoluteValue(0))  >>> 0
```

Functions

Alternative version

```
def absoluteValue(n):
 if n >= 0: return n
 return -n
---
print(absoluteValue(3))  >>> 3
print(absoluteValue(-4)) >>> 4
print(absoluteValue(0))  >>> 0
```

Variable Scope

Variable Scope

A variable declared inside a function is only visible inside the function.

```
def myFunc(x):
    y = 2
    print("happy times!")


print(x)
print(y)
```

Variable Scope

A variable declared inside a function is only visible inside the function.

```
def myFunc(x):
    y = 2
    print("happy times!")

print(x) >>> ERROR!
print(y) >>> ERROR!
```

Variable Scope

If a variable in a function (local variable) has the same name as a variable outside a function (global variable), it will take the value assigned inside the function only inside the function.

```
def otherFunc():
 x = 20
 print(f"x inside the function is {x}")

x = 10
print(f"x before calling the function is {x}")
otherFunc()
print(f"x after calling the function is {x}")
```

Variable Scope

If a variable in a function (local variable) has the same name as a variable outside a function (global variable), it will take the value assigned inside the function only inside the function.

```
def otherFunc():
 x = 20
 print(f"x inside the function is {x}")

x = 10
print(f"x before calling the function is {x}")
otherFunc()
print(f"x after calling the f
```

```
x before calling the function is 10
x inside the function is 20
x after calling the function is 10
```

Variable Scope

If a variable in a function (local variable) has the same name as a variable outside a function (global variable), it will take the value assigned inside the function only inside the function.

```
def otherFunc():
 x = 20
 print(f"x inside the function is {x}")

x = 10
print(f"x before calling the function is {x}")
z = otherFunc()
print(f"x after calling the function is {x}")
```

What is z?

Variable Scope

If a variable in a function (local variable) has the same name as a variable outside a function (global variable), it will take the value assigned inside the function only inside the function.

```
def otherFunc():
 x = 20
 print(f"x inside the function is {x}")

x = 10
print(f"x before calling the function is {x}")
z = otherFunc()
print(f"x after calling the function is {x}")
```

→ **None**

What is z?

Another warning about pointers, memory, is/==...

```
def myBrilliantFunction(l,d):
    print("in the function")
    l[1] = 0
    d["hi"] = "hello"

myList = [1,2,3,4]
myDict = {"a": 100, "b": 200}
print(f"myList before function call: {myList}")
print(f"myDict before function call: {myDict}")
myBrilliantFunction(myList, myDict)
print(f"myList after function call: {myList}")
print(f"myDict after function call: {myDict}")
```

Another warning about pointers, memory, is/==...

```
def myBrilliantFunction(l,d):
    print("in the function")
    l[1] = 0
    d["hi"] = "hello"

myList = [1,2,3,4]
myDict = {"a": 100, "b": 200}
print(f"myList before function call: {myList}")
print(f"myDict before function call: {myDict}")
myBrilliantFunction(myList, myDict)
print(f"myList after function call: {myList}")
print(f"myDict after function call: {myDict}")
```

```
myList before function call: [1, 2, 3, 4]
myDict before function call: {'a': 100, 'b': 200}
in the function
myList after function call: [1, 0, 3, 4]
myDict after function call: {'a': 100, 'b': 200, 'hi': 'hello'}
```

Functions can call themselves (recursion)

Remember the Fibonacci sequence?

```
def fibo(n):
    if n == 0 or n == 1: return 1
    return fibo(n-1)+fibo(n-2)

for x in range(10):
    print(f"{x} - {fibo(x)}")
```

Functions can call themselves (recursion)

Remember the Fibonacci sequence?

```
def fibo(n):
    if n == 0 or n == 1: return 1
    return fibo(n-1)+fibo(n-2)

for x in range(10):
    print(f"{x} - {fibo(x)}")
```

```
0 - 1
1 - 1
2 - 2
3 - 3
4 - 5
5 - 8
6 - 13
7 - 21
8 - 34
9 - 55
```

To be seen in labs

A function to dress up randomly 👖 👗 👕

Functions to better design your game