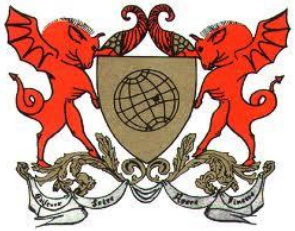
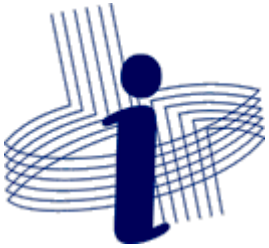


Universidade Federal de Viçosa
Departamento de Informática
Centro de Ciências Exatas e Tecnológicas





# INF 100 – Introduction to Programming

## Functions

## (part 1)

# Motivation

- Calculating $\sqrt{x}$ using the algorithm proposed by Hero of Alexandria (Newton's method):

```
real: x, r
1. read x
2. r <- x/2  // first approximation for the square root
3. r <- (r + x/r) / 2
4. if |r*r - x| > ε, return to step 3
5. ... r contains an approximation for √x
```

where $\varepsilon$ = a very small value, for instance, $10^{-5}$.

# Motivation

- Possible implementation in Python:

```
x = float( input('Type the value of x: '))
r = x/2  # initial value for root calculation
while abs( r*r - x ) > 1e-10:
    r = (r + x/r) / 2
print('Square root of', x, 'is', r )
```


# Motivation

- Suppose we were required to use this method every time we needed to calculate a square root.
  - The code is not clear.
  - The code is difficult to be reused in other programs.
  - It is *error-prone* (tending to cause erros).
  - Why is it necessary to know details of how to calculate a square root every time it is needed?


# Functions

- These aspects are some reasons for using functions.
- Functions are named sections of a program that perform a specific task:

```
import math

x = float( input('Type the value of x: '))
r = math.sqrt( x )
print('Square root of', x, 'is', r )
```


# Functions in Python

- Some functions from standard library `math`:
  - `log(x)`
  - `log10(x)`
  - `exp(x)`
  - `sqrt(x)`
  - `tan(x)`
  - `sin(x)`
  - `cos(x)`
  - …


Identifying possibilities for code reuse...

```
...

while True:
    n = int( input("Type the number of students: "))
    if n < 2 or n > 50:
        print("Value must be between 2 and 50")
    else:
        break
sum = 0
for i in range(0, n):
    while True:
        s = "Type the grade for student " + str(i+1+1) + ": "
        x = int( input(s) )
        if x < 0 or n > 100:
            print("Value must be between 0 and 100")
        else:
            break
    sum += x
average = sum / n
```

Identifying possibilities for code reuse...

```
...

while True:
    n = int( input("Type the number of students: "))
    if n < 2 or n > 50:
        print("Value must be between 2 and 50")
    else:
        break
sum = 0
for i in range(0, n):
    while True:
        s = "Type the grade for student " + str(i+1+1) + ": "
        x = int( input(s) )
        if x < 0 or n > 100:
            print("Value must be between 0 and 100")
        else:
            break
    sum += x
average = sum / n
```

# Function in Python

- Suppose a function `readvalue` is available.
- Parameters: a *message*, the *minimum* and *maximum* values allowed.


# Function in Python

- Suppose a function `readvalue` is available.
- Parameters: a *message*, the *minimum* and *maximum* values allowed.

```
n = readValue("Type number of students: ", 2, 50)

sum = 0
for i in range(0, n):
    s = "Type the grade for student " + str(i+1+1) + ": "
    x = readValue(s, 0, 100)
    sum += x
average = sum / n

print("The average is", average)
```

(The code for `readvalue` will be discussed later)


# Functions

- Advantages of using functions:
  - Modularity and clarity of code.
  - Easiness for code reuse.
  - Decreases the chances of errors.
  - Separates the use of a functionality from the details of its implementation.
  - Allows reasoning about an algorithm in a higher level and helps stepwise refinement of the code.


# Functions in Python

- Steps for creating and using functions:
  - Declare a function and define its code
  - Call the function from other parts of the program


# Functions in Python

- In order to create a new function, it is important to know:
  - If parameters (input data) are necessary, what are these parameters?
  - If the function returns values, what are these values?
- This information will define how the function will be designed.


# Definition of functions in Python

```
def name( list of parameters ):
    commands
    return value(s) to be returned
```

- **`name`**: name used to call (use) a function.
- **`list of parameters`**: data passed to the function (names separated by commas).


# Example – function **readValue**

```
def readValue( msg, min, max ):
    while (True):
        v = int( input( msg ))
        if v < min or v > max:
            print("Value must bebetween", min, "and", max )
        else:
            break
    return v


n = readValue("Type number of students: ", 2, 50)

sum = 0
for i in range(0, n):
    s = "Type the grade for student " + str(i+1+1) + ": "
    x = readValue(s, 0, 100)
    sum += x
average = sum / n

print("The average is", average)
```

# Exercise

- Create a function named `abs` that accepts as parameter a value $x$ and returns the absolute value of $x$.


# Solution

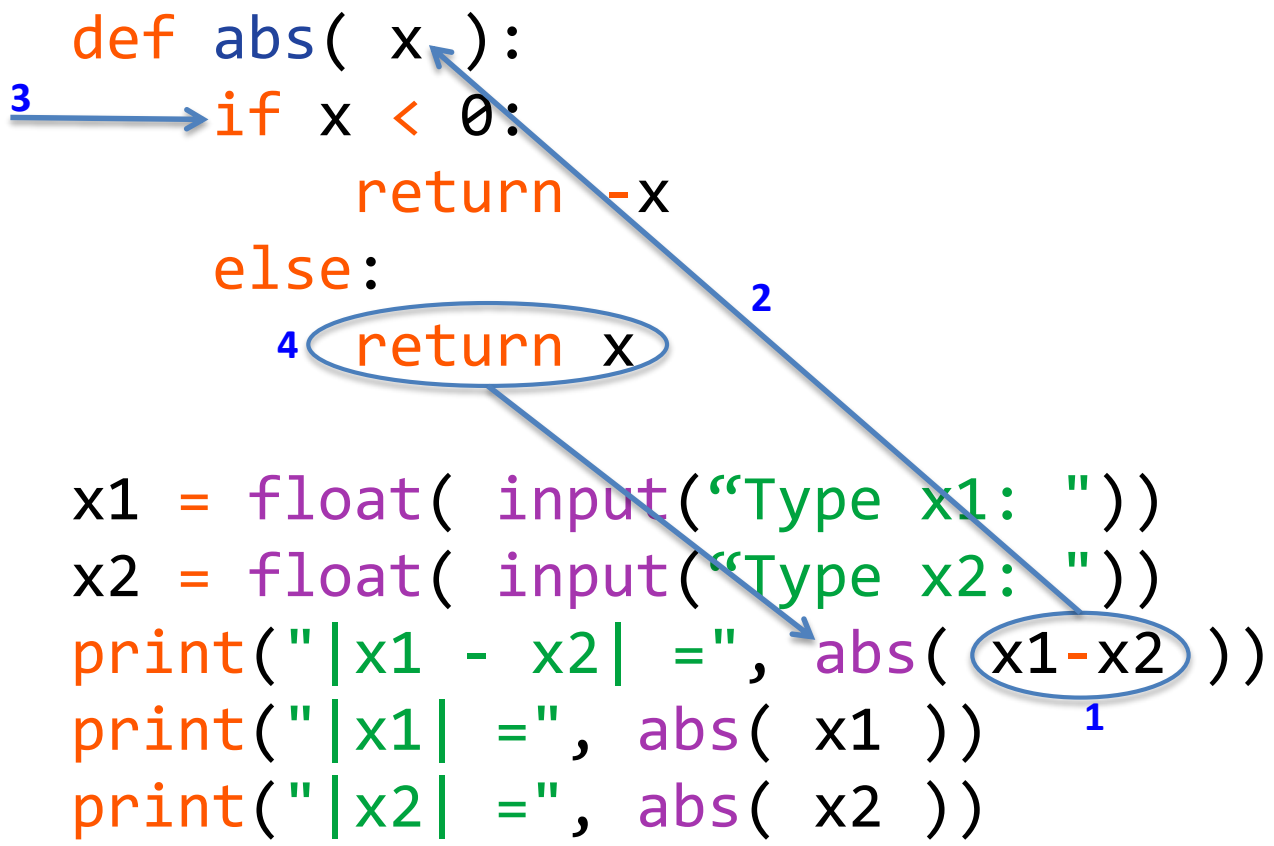
```
def abs( x ):
    if x < 0:
        return -x
    else:
        return x
```


# Using the function in a program

```
def abs( x ):
    if x < 0:
        return -x
    else:
        return x

x1 = float( input("Type x1: "))
x2 = float( input("Type x2: "))
print("|x1 - x2| =", abs( x1-x2 ))
print("|x1| =", abs( x1 ))
print("|x2| =", abs( x2 ))
```


# Using the function in a program



# Passing parameters by value

1. The **arguments** of the functions are expressions which are evaluated at the place of call.
2. The result of this evaluation is copied to the formal parameters of the function, following the order of declaration (the first argument is copied to the first parameter, and so on).
3. The flow of execution jumps to the first statement of the function, executing the code of the function.
4. The execution of the function finishes when its last command is executed, or when a ***return*** command is performed. The flow of execution returns to the point of the call, carrying the value of result (if it exists).


# Functions in Python

- The function *abs* returns a floating point value.
- It is also possible to create functions that execute a specific action but return no value as result.

  Example:

```
Beep( freq, time_ms )
```

  Purpose: make a sound given a specific frequence and a time, in milliseconds .

# Functions in Python

```
def Beep( freq, time_ms ):
    ...
    ...
    ...
```

In this case, a ***return*** command is not necessary. The execution of the function can finish when the last statement of the function is executed.

```
def Beep( freq, time_ms ):
    ...
    ...
    return
```

**OR** it is possible to use a return command with no expression. following it.


# Functions in Python

- It is also possible to have functions with no parameters. Examples:

```
def MessageBeep():
    ...

def pi():
    return 3.14159265358979
```


# Functions in Python

- Examples of use:

```
MessageBeep()
Beep( 200, 1000 )
print( pi() )
```


# Other examples

```
...
def average( x, y ):
    a = (x + y) / 2
    return a

z = average( 5.5, 7.8 )
print('Average =', z )
```


# Other examples

```
...
def average( x, y ):
    return (x + y) / 2


z = average( 5.5, 7.8 )
print('Average =', z )
```


# Other examples

```
...
def largest( a, b):
    if a > b:
        return a
    else:
        return b

x = float( input('x = '))
y = float( input('y = '))
print('Largest value =', largest(x, y) )
```


# Common mistakes

```
def f( x, y ):
    r = x*x + y
```

(forget *return*)

```
def myFunc( x, y ):
    return x*x + y

def pi():
    return 3.1415926535897

print( pi )
print( myFunc( 12 ))
```

pi()

(parameter missing)

# Exercises

- Create a function that receives as parameters 3 values and returns the largest value.

- Create a function that receives as parameter an integer value $x$ and returns $x!$.