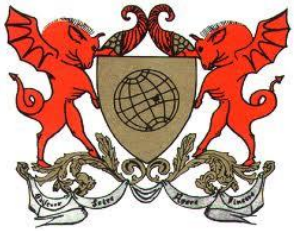
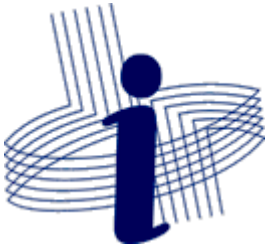


Universidade Federal de Viçosa
Departamento de Informática
Centro de Ciências Exatas e Tecnológicas



INF 100 – Introduction to Programming

Functions

(cont.)

Scope of Variables

Local variables

```
def f( x, y ):
    k = 2*x + y
    return k

z = 2
w = 1
print( f( 2*z, w ))
```

Scope of Variables

Local variables

```
def f( x, y ):
    k = 2*x + y
    return k

z = 2
w = 1
print( f( 2*z, w ))
```

x, **y** and **k** are variables "visible" only **inside** function **f()**. They can only be used inside f().
So they are called **local variables**.

z and **w** are variables defined in the main program.
In principle, they should only be used in the main program.


Scope of Variables

Local variables

```
def f( x, y ):
    z = 2*x + y
    return z

z = 2
w = 1
print( z )
print( f( 2*z, w ))
print( z )
```


Scope of Variables

Local variables

```
def f( x, y ):
    z = 2*x + y
    return z

z = 2
w = 1
print( z )
print( f( 2*z, w ))
print( z )
```

The variables with name **z** in function **f()** and in the main program are considered as two **distinct variables**! They are in diferente scopes (for this reason, they **can have a same name**).

Any variable which is changed or created inside of a function is local


Example: function printMatrix

```
def printMatrix(M):
    lines, columns = M.shape
    for i in range(0, lines):
        for j in range(0, columns):
            print('%5.1f' % (M[i][j]), end='')
        print()
```

The code of function **printMatrix** involves only LOCAL variables!

```
m = int( input('Number of lines in matrix A: '))
n = int( input('Number of columns in matrix A: '))
p = int( input('Number of columns in matrix B: '))

print("\nreading matrix A...")
A = readMatrix(m, n)

print('\nMatrix A:')
printMatrix(A)

print("\nreading matrix B...")
B = readMatrix(n, p)

print('\nMatrix B:')
printMatrix(B)

C = calcProduct(A, B)

print('\nProduct AB :')
printMatrix(C)
```

The function **printMatrix** is called 3 times in this program, using different arguments.

Scope of Variables

Local variables

```
def f( x, y ):
    k = 2*x + y + z
    return k

z = 2
w = 1
print( f( 2*z, w ))
```

Scope of Variables

Local variables

```
def f( x, y ):
    k = 2*x + y + z
    return k
```

Local scope of **f()**

In this case, the variable **z** in **f()** is considered as a **global variable**, because it was not created inside the scope of **f()**, but before **f()** is called.

```
z = 2
w = 1
print( f( 2*z, w ))
```

Local scope of the main program

Any variable which is changed or created inside of a function is local


Example with global variable

```
def printMatrix():
    lines, columns = M.shape
    for i in range(0, lines):
        for j in range(0, columns):
            print('%5.1f' % (M[i][j]), end='')
        print()
```

In the code above, the function printMatrix has no parameters. What is the difference between this version and the previous one?


Scope of Variables

Local variables

```
def f( x, y ):
    k = 2*x + y + z
    z = 1
    return k

z = 2
w = 1
print( f( 2*z, w ))
```


Scope of Variables

Local variables

```
def f( x, y ):
    k = 2*x + y + z
    z = 1
    return k

z = 2
w = 1
print( f( 2*z, w ))
```

This example produces an error, because there is an ambiguity. Variable **z** should be considered as a local or a global variable?

Any variable which is changed or created inside of a function is local

Scope of Variables

Local variables

```
def f( x, y ):
    k = 2*x + y + z
    global temp
    temp = 2
    return k

temp = 0
z = 2
w = 1
print( temp )
print( f( 2*z, w ))
print( z, w )
print( temp )
```


Scope of Variables

Local variables

```
def f( x, y ):
    k = 2*x + y + z
    global temp
    temp = 2
    return k
```

Local scope of f()

Global variable (not recommended)

```
temp = 0
z = 2
w = 1
print( temp )
print( f( 2*z, w ))
print( z, w )
print( temp )
```

Local scope of the main program

What will be printed?

Any variable which is changed or created inside of a function is local, <u>if it hasn't been declared as a global variable.</u>

Scope of Variables

Local variables

```
def f( x, y ):
    k = 2*x + y + z
    global temp
    temp = 2
    return k

temp = 0
z = 2
w = 1
print( temp )
# print( f( 2*z, w ))
print( z, w )
print( temp )
```


Scope of Variables

Local variables

```
def f( x, y ):
    k = 2*x + y + z
    global temp
    temp = 2
    return k
```

Local scope of f()

Global variable (not recommended)

```
temp = 0
z = 2
w = 1
print( temp )
# print( f( 2*z, w ))
print( z, w )
print( temp )
```

Local scope of the main program

What will be printed?

Example – revisiting Minesweeper

- In the game **Minesweeper**, a certain number of bombs is distributed in a map, represented by a matrix. A cell that does not contain a bomb must indicate the number of bombs it has around it.
- Write a program that reads the size of the map and the number of bombs. Then it must distribute the bombs randomly and then calculate the number of bombs around each non-bomb cell.


Sample execution
(suppose that a bomb is represented by “9”)

```
Size of the square field: 4
Number of bombs: 5

Distributing bombs:
 0 0 0 0
 0 9 9 0
 9 9 0 0
 0 0 9 0

Calculating bombs around each cell:
 1 2 2 1
 3 9 9 1
 9 9 4 2
 2 3 9 1
```

Solution (1/3)

```
bomb = 9

while True:
    n = int(input("Size of the square map: "))
    if n < 3 or n > 10:
        print("Value must be between 3 and 10")
    else:
        break

maxb = math.ceil(0.3*n*n)
print("Maximum number of bombs (30%) =", maxb)

while True:
    b = int(input("Number of bombs: "))
    if b < 1 or b > maxb:
        print("Value must be between 1 and", maxb)
    else:
        break

map = np.zeros((n, n), dtype=int)
```


Solution (2/3)

```
k = b
while k > 0:
    i = random.randint(0, n-1)
    j = random.randint(0, n-1)
    if map[i][j] == 0:
        map[i][j] = bomb
        k -= 1

print()
for i in range(0, n):
    for j in range(0, n):
        print('%2d' % (map[i][j]), end='')
    print()
```


Solution (3/3)

```
for i in range(0, n):
    for j in range(0, n):
        if map[i][j] == bomb:
            if i-1 >= 0 and j-1 >= 0 and i-1 < n and j-1 < n and map[i-1][j-1] != bomb:
                map[i-1][j-1] += 1
            if i-1 >= 0 and j >= 0 and i-1 < n and j < n and map[i-1][j] != bomb:
                map[i-1][j] += 1
            if i-1 >= 0 and j+1 >= 0 and i-1 < n and j+1 < n and map[i-1][j+1] != bomb:
                map[i-1][j+1] += 1
            if i >= 0 and j-1 >= 0 and i < n and j-1 < n and map[i][j-1] != bomb:
                map[i][j-1] += 1
            if i >= 0 and j+1 >= 0 and i < n and j+1 < n and map[i][j+1] != bomb:
                map[i][j+1] += 1
            if i+1 >= 0 and j-1 >= 0 and i+1 < n and j-1 < n and map[i+1][j-1] != bomb:
                map[i+1][j-1] += 1
            if i+1 >= 0 and j >= 0 and i+1 < n and j < n and map[i+1][j] != bomb:
                map[i+1][j] += 1
            if i+1 >= 0 and j+1 >= 0 and i+1 < n and j+1 < n and map[i+1][j+1] != bomb:
                map[i+1][j+1] += 1

print()
for i in range(0, n):
    for j in range(0, n):
        print('%2d' % (map[i][j]), end='')
    print()
```


Exercise – revisiting Minesweeper

Rebuild the program using the functions:

- **readValue**: given a msg and an interval, read a value from the input ensuring the value lies inside the given interval
- **printMap**: print the map on the screen
- **inc**: given a line x and a column y, checks if it represents a valid position of the map and if there is not a bomb there; if so, increment the number stored in that cell.


Solution (1/2)

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

def printMap():
    for i in range(0, n):
        for j in range(0, n):
            print('%2d' % (map[i][j]), end='')
        print()

def inc(x, y):
    if x >= 0 and x < n and y >= 0 and y < n and map[x][y] != bomb:
        map[x][y] += 1
```

Solution (2/2)

```
bomb = 9

n = readValue("Size of the square map: ", 3, 10)
maxb = math.ceil(0.3*n*n)
print("Maximum number of bombs (30%) =", maxb)
b = readValue("Number of bombs: ", 1, maxb)

map = np.zeros((n, n), dtype=int)
k = b
while k > 0:
    i = random.randint(0, n-1)
    j = random.randint(0, n-1)
    if map[i][j] == 0:
        map[i][j] = bomb
        k -= 1

print()
printMap()

for i in range(0, n):
    for j in range(0, n):
        if map[i][j] == bomb:
            inc(i-1, j-1)
            inc(i-1, j)
            inc(i-1, j+1)
            inc(i, j-1)
            inc(i, j+1)
            inc(i+1, j-1)
            inc(i+1, j)
            inc(i+1, j+1)

print()
printMap()
```

Returning more than one result

```
def sort( x, y ):
    if (x <= y):
        return x, y
    else:
        return y, x

a = float( input('Type a number: '))
b = float( input('Type another number: '))
a, b = sort( a, b )
print( a, '<', b )
```


Exercise 1

- Using function **sort()** as defined in the previous slide, write a program that reads 3 integer values and prints these values in ascending order.

  Example:

  -Input: `A=7 B=3 C=5`

  -Output on the screen: `A=3 B=5 C=7`


Summary – scope and parmeters

- Global variables: visible in all functions of the program;
- Local variables: visible only inside the functions they are defined;
- Function parameters are considered as local variables:
  - Parameters of simple types as *integer* or *floating point* are passed **by value**. It means they are **independent** from variables, expressions etc. which generated the values that were passed to the function.


Passing arrays as parameters

- When an array is passed as a parameter to a function, a copy of the array is **not** generated (unlike integer or floating point parameters). Copying arrays would be expensive in terms of time and memory space.
- When an array is passed as a parameter to a function, only the **address** of the array is passed.
- Consequence: any modification on the array (passed as parameter) inside a function produces a modification on the variable that was passed as a parameter.


Passing arrays as parameters

- Example:

```
def double( vector ):
    n = len( vector )
    for i in range( 0, n ):
        vector[i] *= 2

a = [4, 3, 2, 1]
print( a )
double( a )
print( a )
```


Exercise 2

- Write a function that receives as parameters a string *s* and a character *c*, and returns the number of occurrences of *c* in *s*.


Exercise 3

- Write a function that receives an array of floating point numbers and returns the average and the standard deviation of the numbers.

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n} \qquad s = \sqrt{\frac{\sum_{i=1}^{n} \left(x_i - \bar{x}\right)^2}{n}}$$

Exercise 4

- Write a function `CPF_digits()` that receives a CPF number with format xxxxxxxxx (without checking digits), and returns a string with the 2 checking digits.

- Here you can find how CPF checking digits are calculated:

  http://www.geradorcpf.com/algoritmo_do_cpf.htm


Exercise 5

- Using the function of Exercise 4, write a function `CPF_valid()` that receives a complete CPF number in the format xxxxxxxxxyy, xxxxxxxxx-yy or xxx.xxx.xxx-yy. The function must return True if the string represents a valid CPF number, and False otherwise.
- Tip: write an auxiliary function that filters only the digits from a given string. You can use this function in the user input, producing a string in a simple format, before processing it.


Turtles in Python

```
import turtle as t

def poligono_regular( n, tamanho ):
    # Calcular o complemento do ângulo interno do polígono
    ang_interno = 180 - (n-2)*180/n
    for i in range(0, n):
        t.forward( tamanho )
        t.right( ang_interno )

while True:
    n = int(input('\nNúmero de lados do polígono: '))
    if n < 3: break
    tam = int(input('Comprimento de cada lado: '))
    poligono_regular( n, tam )
```


```
# Desenhar quadrados concêntricos

import turtle as t

def quadrado( tamanho ):
    for i in range(0, 4):
        t.forward( tamanho )
        t.right( 90 )

def shift( delta_x, delta_y ):
    t.up()
    t.goto( t.xcor() + delta_x, t.ycor() + delta_y )
    t.down()

h = int(input('Tamanho do quadrado externo: '))

while h > 0:
    quadrado( h )
    shift( 5, -5 )
    h -= 10

t.Screen().exitonclick()
```