SEE1002
Introduction to Computing for Energy and Environment

Part 2: Elements of Python programming

Sec. 5: Functions

Course Outline

Outline

1. Motivation
2. How are functions used?
3. Functions in Flowgorithm
4. Implementation in Python

1. Motivation

Flow control

For the past few weeks, we have discussed methods for organising program flow:

- `sequential flow` *perform tasks one after the other*
- `if-else` *decision making*
- `while, for` *repeating tasks*

- This is enough to write any program!
- However, this does not guarantee that we will be able to write a good program.

Elements of a good program

What constitutes a good program? To a large extent the criteria are no different from those applied to any kind of writing:

1. Does what it's supposed to do (*relevance*)
2. Does it properly (*correctness*)
3. Doesn't waste time doing unnecessary things *(efficiency)*
4. Isn't longer than necessary *(conciseness)*
5. Is easy to follow *(readability)*

Code reuse

Code reuse

The need for code reuse arises because sequences of instructions often appear over and over.

- Repeating these instructions blindly makes the code **longer than necessary**.
- It also makes the code **hard to read.**

The objective of code reuse is to avoid repeating these statements in full. *But how?*

Ways of reusing code

For our purposes, there are basically two ways of reusing code:

1. loops — set of instructions is executed *sequentially* or one after another. *Already covered!*

2. functions — set of instructions is executed *intermittently* (i.e. from time to time) or upon request.

What is a function?

A function is analogous to the formula function of a calculator or a mathematical function.

- `x → f(x)`
- `x,y → g(x,y)`
- `etc.`

Key idea: each input value (or values) is uniquely mapped to an output value (or values).

Differences between functions in mathematics and Python

A function in a computer language such as Python is more general than a mathematical function.

- The function can do more than return a value (e.g. print intermediate values)
- In fact, the function does not necessarily return a value.

- In some computer languages (e.g. C or FORTRAN), one distinguishes between functions that return a value and ones that do not, i.e. they have different names. ***This is not done in Python.***

2. How are functions used?

Benefits of using functions

We have already explained that functions increase code reuse leading to

1. Shorter code

2. More readable code

More generally, functions promote structured programming

What is structured programming?

Structured programming refers to a type of program in which:

1. Program is easily decomposed into distinct components or blocks.
2. Program flow proceeds logically from one block to another.
3. Blocks can be connected via sequential flow or functions.

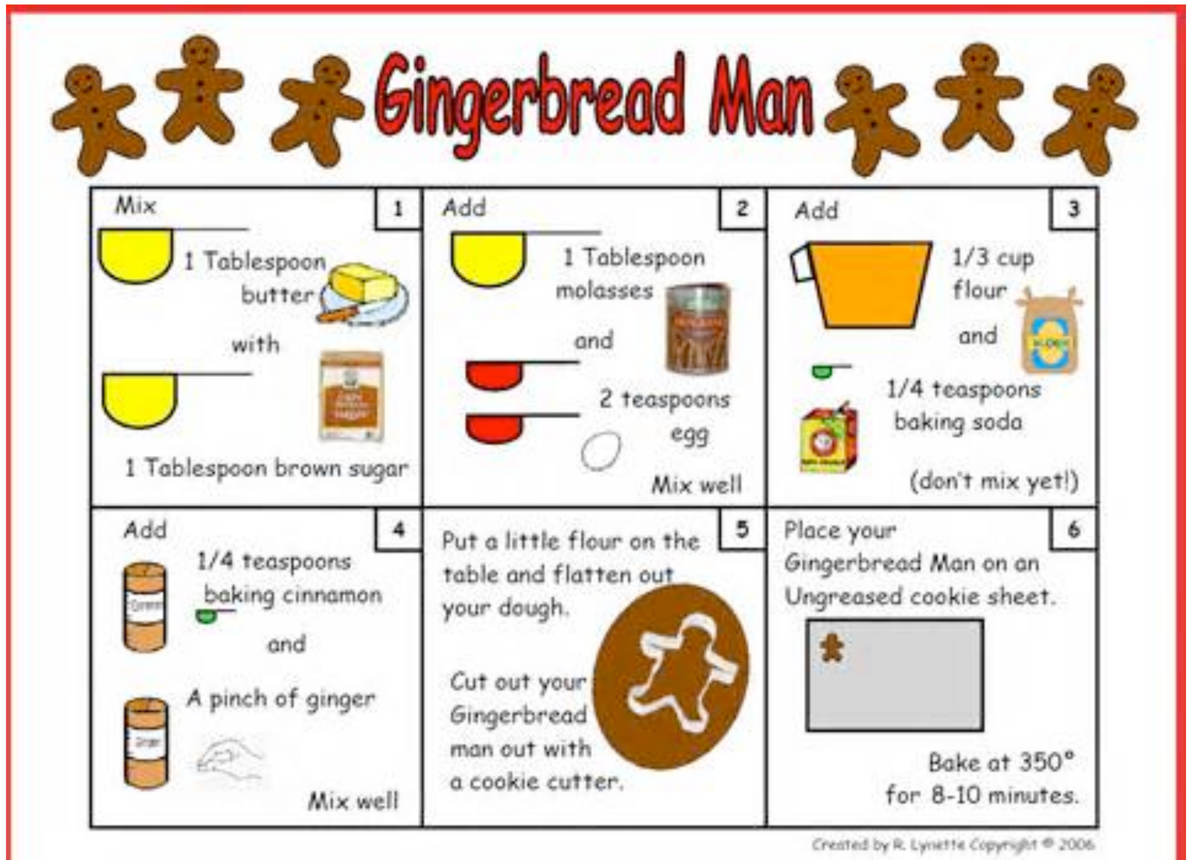
What is a code block?

- Roughly speaking a code block is a group of statements or instructions that accomplish a specific task.
- *Examples:*
  - User prompt (e.g. `Y/N/Q`)
  - Printing a specified range of numbers (e.g. `1-100`)
  - Calculating the average of all the numbers in a list (e.g. `[`x_1`,` x_2…x_N`]`)
- In many cases ***a code block can be turned into a function.***

How to define a code block?

- At the beginning of this course we mentioned that a recipe is analogous to a program.
- In any recipe one can't describe each step in complete detail.

A code block corresponds to a step in a recipe.



Gingerbread Man

Mix 1 1 Tablespoon butter with 1 Tablespoon brown sugar	Add 2 1 Tablespoon molasses and 2 teaspoons egg Mix well	Add 3 1/3 cup flour and 1/4 teaspoons baking soda (don't mix yet!)
Add 4 1/4 teaspoons baking cinnamon and A pinch of ginger Mix well	Put a little flour on the table and flatten out your dough. 5 Cut out your Gingerbread man out with a cookie cutter.	Place your Gingerbread Man on an Ungreased cookie sheet. 6 Bake at 350° for 8-10 minutes.

Created by R. Lynette Copyright © 2006

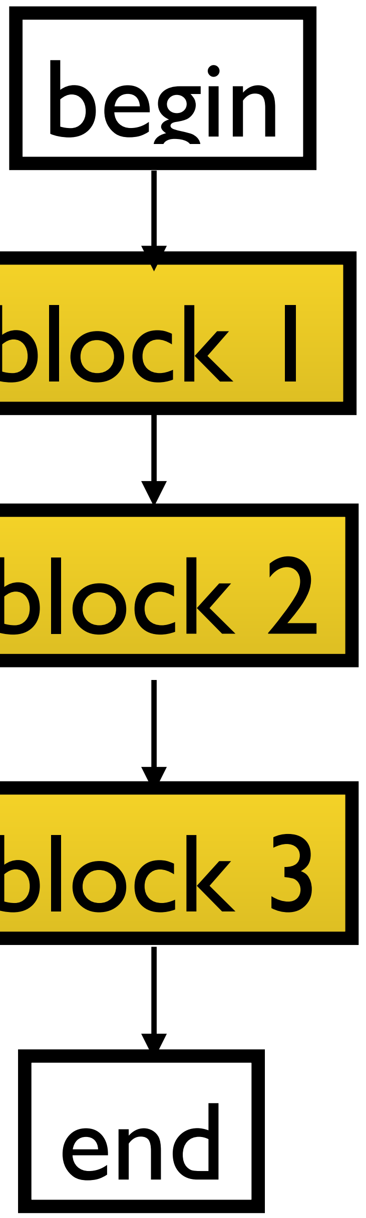
Structured programming illustrated

We now want to illustrate the idea of structured programming using different kinds of program flow or flow charts.

1. Purely linear or sequential flow.
2. Code reuse via loops
3. Code reuse via function calls
4. Poorly structured code

1. Sequential flow revisited

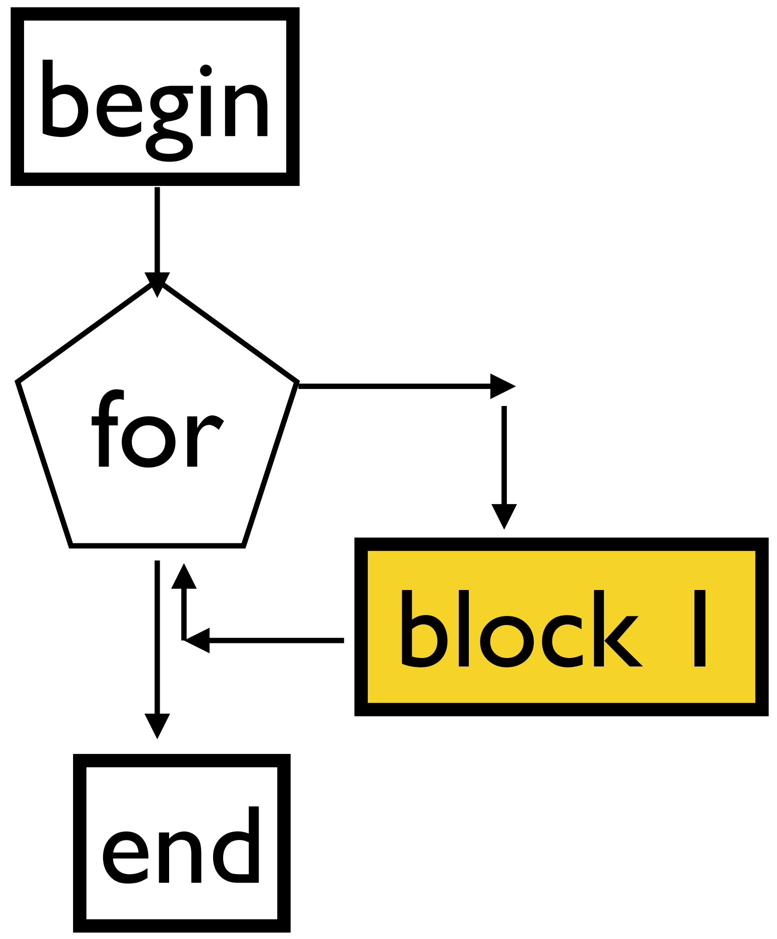
The simplest programs involve a sequence of blocks. that are always executed in the same order. The program is well-structured, i.e., the flow is easy to follow if there aren't too many blocks:



N.B. each block can contain more than one line!

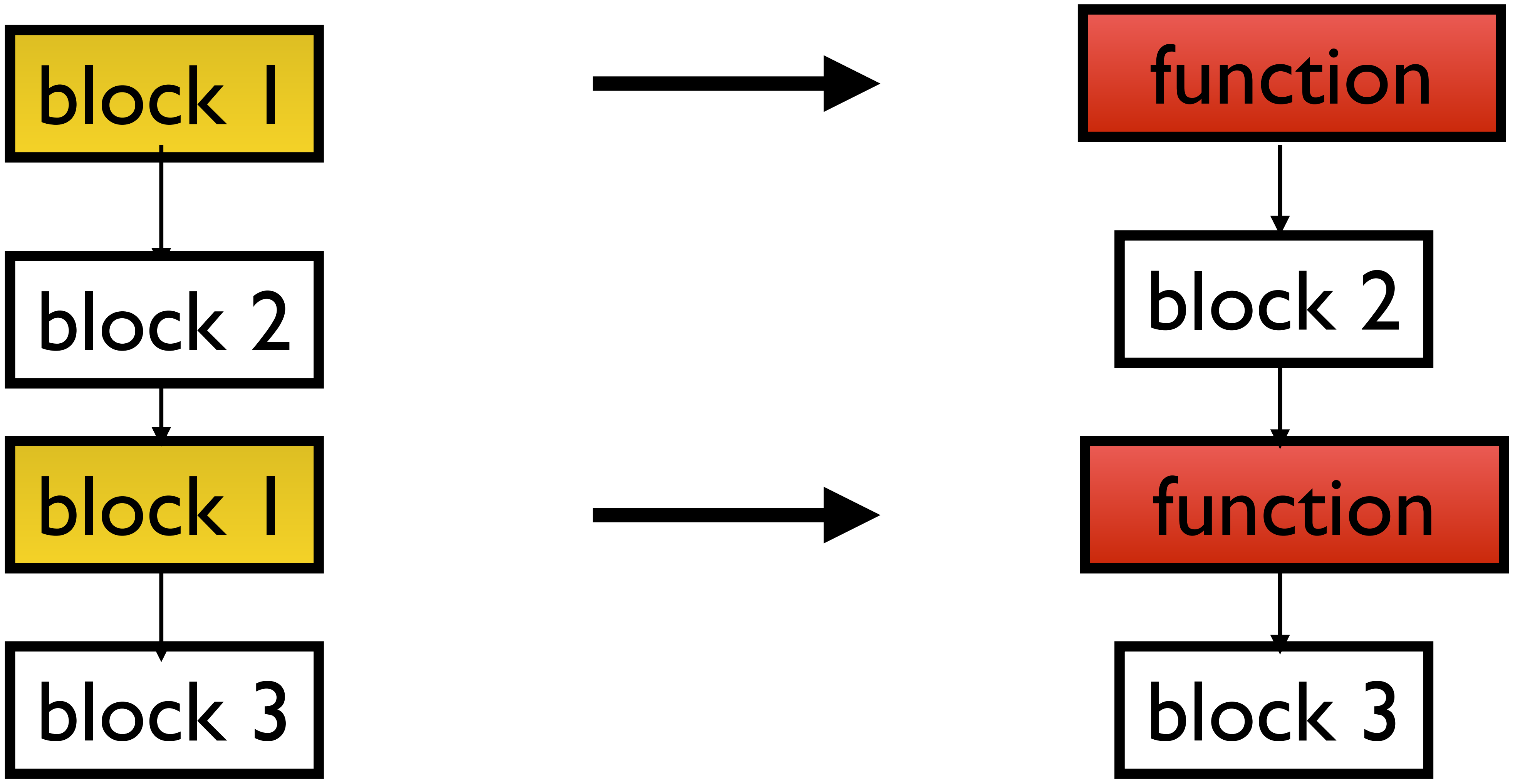
2. Code reuse using loops

The problem with purely sequential flow is that it doesn't work well if there are too many blocks. If the blocks are identical then the structure of the code can be made more transparent with a loop.



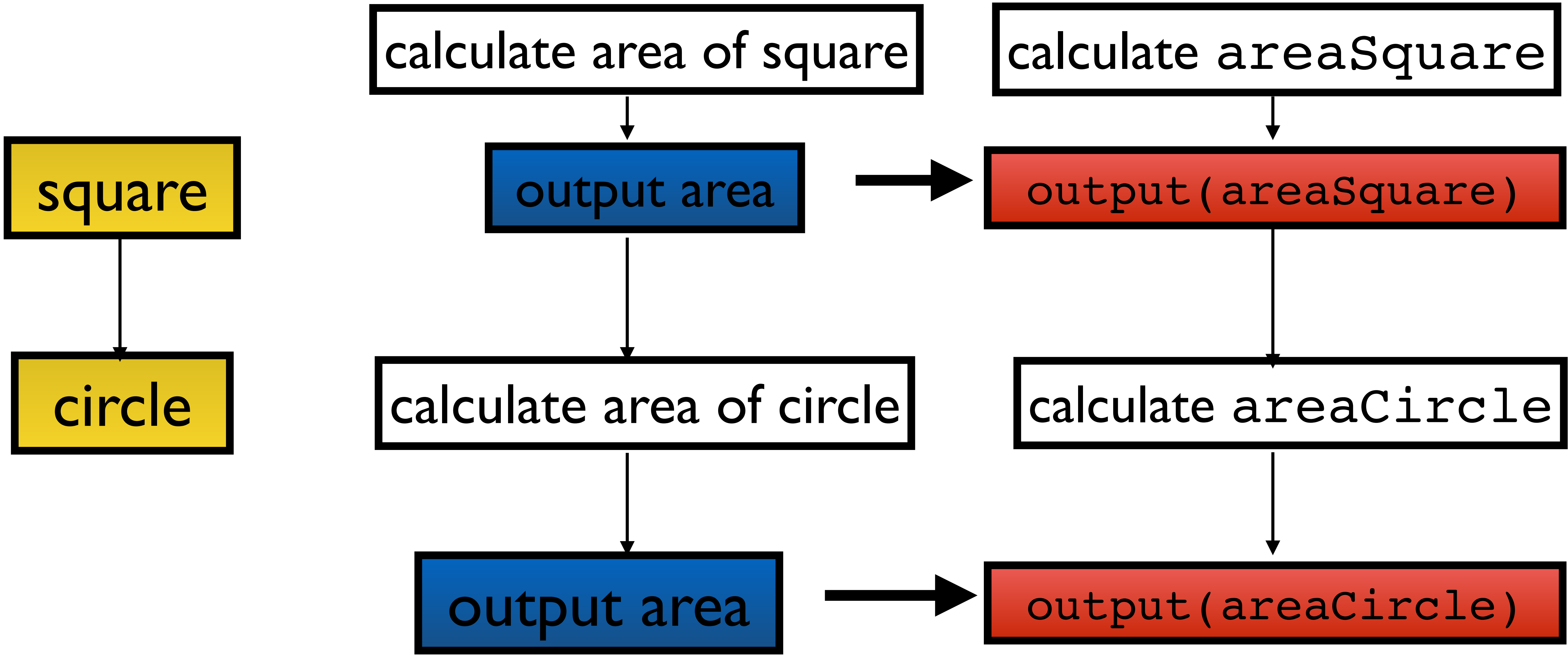
3. Code reuse using functions

Sometimes we need to reuse code that is required intermittently instead of sequentially (as in the case of a loop). This can be done with functions.



Illustration

Consider a program that calculates and outputs the area of a square and a circle.



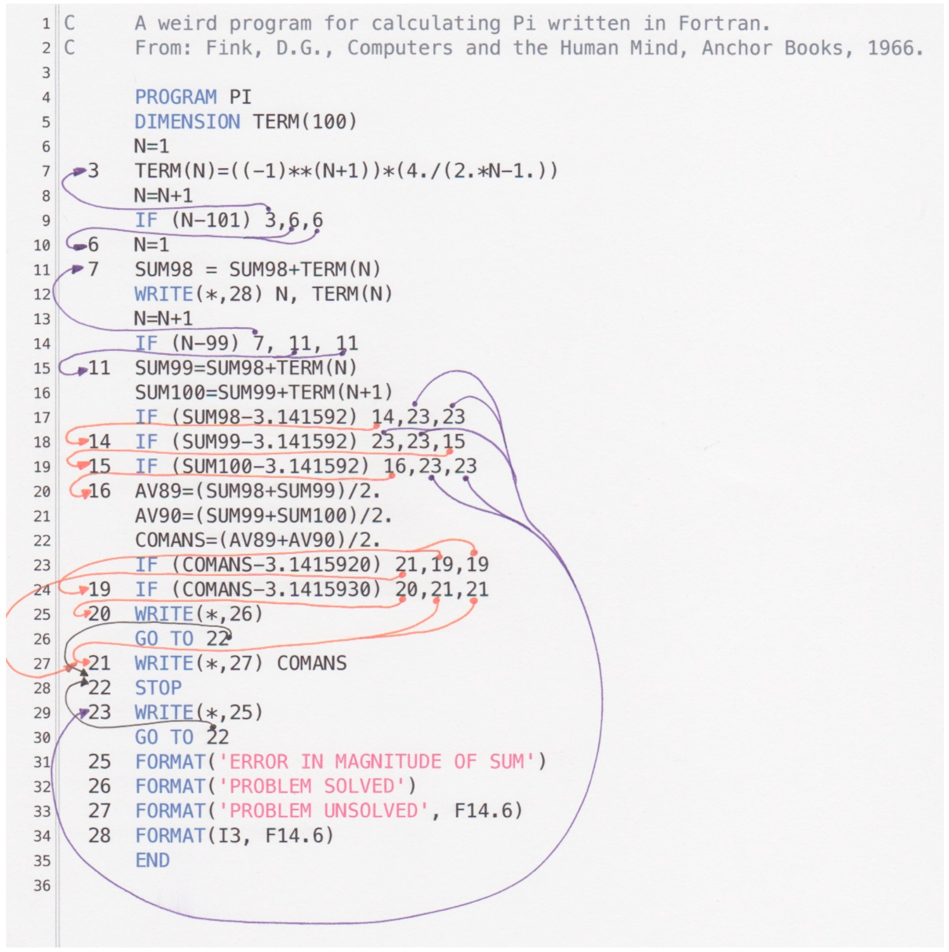
4. Poorly structured code

A poorly structured program is one in which the flow is not clear. This can be caused by:

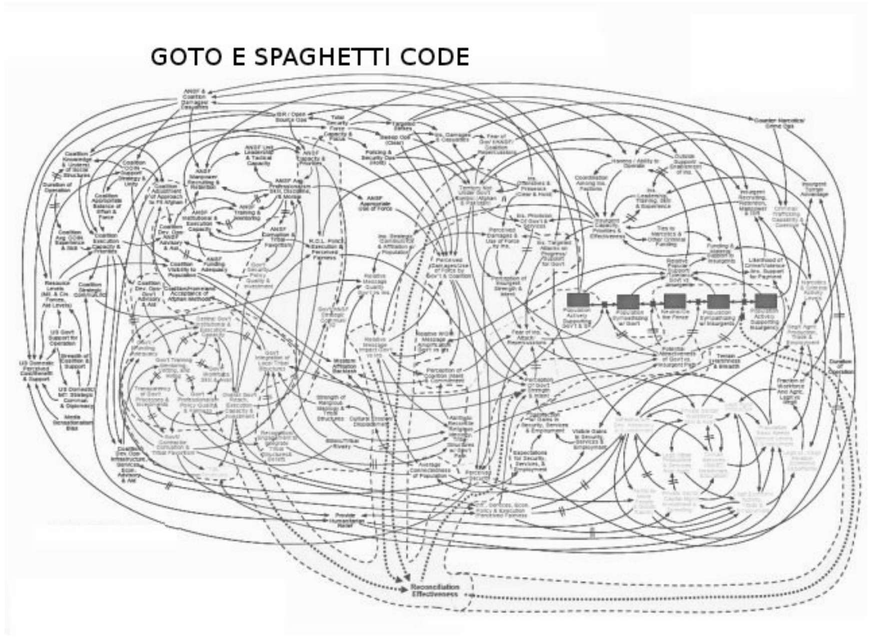
1. Many `if` tests
2. Long and complicated blocks
3. Use of `goto` (not possible in Python!)

Spaghetti code

Jumping from one part of a program makes the program flow very hard to follow. This can lead to spaghetti code!



```
C     A weird program for calculating Pi written in Fortran.
C     From: Fink, D.G., Computers and the Human Mind, Anchor Books, 1966.

      PROGRAM PI
      DIMENSION TERM(100)
      N=1
 3    TERM(N)=((-1)**(N+1))*(4./(2.*N-1.))
      N=N+1
      IF (N-101) 3,6,6
 6    N=1
 7    SUM98 = SUM98+TERM(N)
      WRITE(*,28) N, TERM(N)
      N=N+1
      IF (N-99) 7, 11, 11
11    SUM99=SUM98+TERM(N)
      SUM100=SUM99+TERM(N+1)
      IF (SUM98-3.141592) 14,23,23
14    IF (SUM99-3.141592) 23,23,15
15    IF (SUM100-3.141592) 16,23,23
16    AV89=(SUM98+SUM99)/2.
      AV90=(SUM99+SUM100)/2.
      COMANS=(AV89+AV90)/2.
      IF (COMANS-3.1415920) 21,19,19
19    IF (COMANS-3.1415930) 20,21,21
20    WRITE(*,26)
      GO TO 22
21    WRITE(*,27) COMANS
22    STOP
23    WRITE(*,25)
      GO TO 22
25    FORMAT('ERROR IN MAGNITUDE OF SUM')
26    FORMAT('PROBLEM SOLVED')
27    FORMAT('PROBLEM UNSOLVED', F14.6)
28    FORMAT(I3, F14.6)
      END
```



In spaghetti code, the relations between the pieces of code are so tangled that it is nearly impossible to add or change something without unpredictably breaking something somewhere else.

How do functions help?

To improve code structure, functions can be used for:

1. Long and complicated blocks
2. Instructions that are frequently repeated
3. Combining related steps into a single block

In a nutshell, functions can be used for well-defined tasks.

Application of structured programming

In structured programming we to ***reduce a program to a sequence of blocks***. In principle, however, each of these blocks can be reduced further using loops and functions. *So when do we stop?*

- This is usually a matter of judgment and experience. Real programs rarely look as nice as the simple examples we've been considering.
- One can get carried away but the idea of simplifying the program structure by using functions is extremely valuable.

3. Functions in Flowgorithm

Motivation

- Before considering functions in Python, it's useful to review how they're used in Flowgorithm.
- Some of the details differ in Python, but the basic concepts are the same.

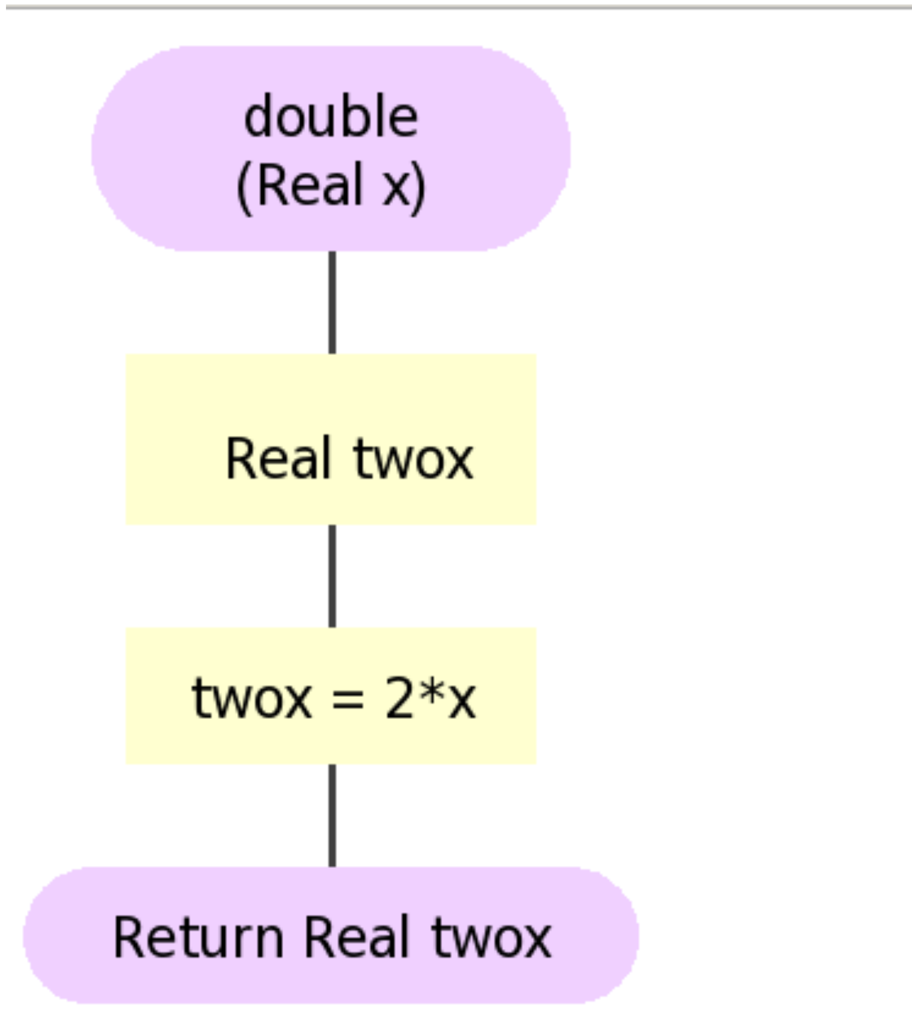
How is a function defined?

- In Flowgorithm, a function is defined by:
  - name
  - arguments
  - return value
- The same is basically true for other computer languages.

Example from Flowgorithm

For example, in the function `double(x)`:

- the function name is `double`
- the argument is the real variable `x`
- the return value is a real variable



Comments

1. A function does not necessarily have a return value. In other languages such functions have a special name (e.g. `procedure` or `subroutine` or `void`).

2. A function does not necessarily have any arguments.

    - This is uncommon but occasionally useful.

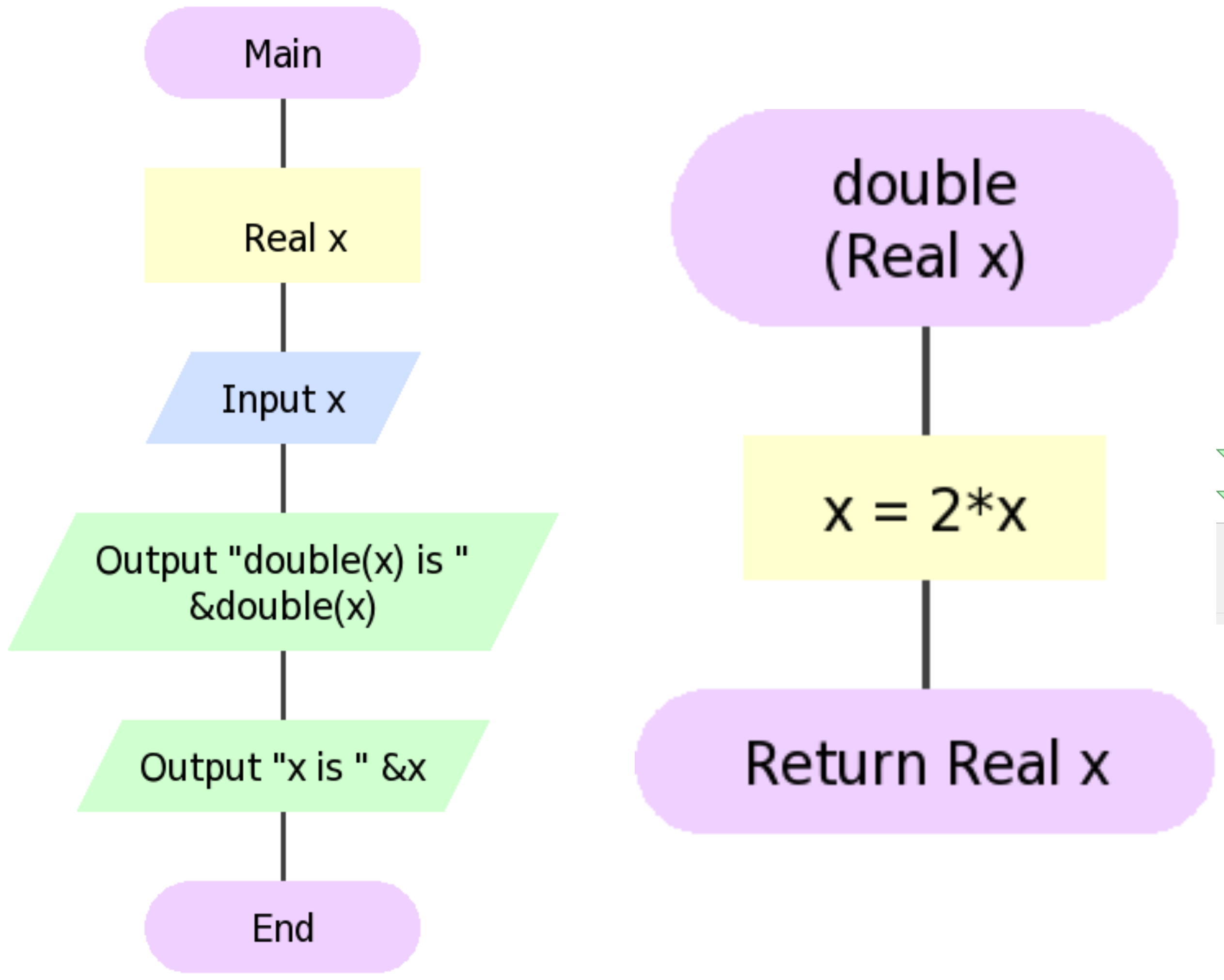
    - *Example:* `sys.exit()`

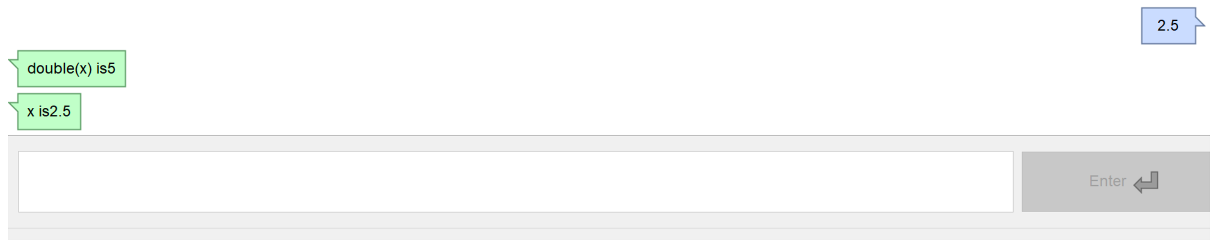
What about arguments?

The treatment of arguments is trickier. *Can they be modified by a function?*

- In most languages, modifications within a function are ***limited to the function,*** i.e., they do not affect the original variable. This is referred to a passing by value.
- In some languages, ***changes within the function can affect the original variable***. This is referred to as passing by reference.

Example from Flowgorithm





```
                                                    2.5
double(x) is5
x is2.5
```

Modifications to `x` within the function do not affect `x` outside the function. Hence Flowgorithm follows pass by value.

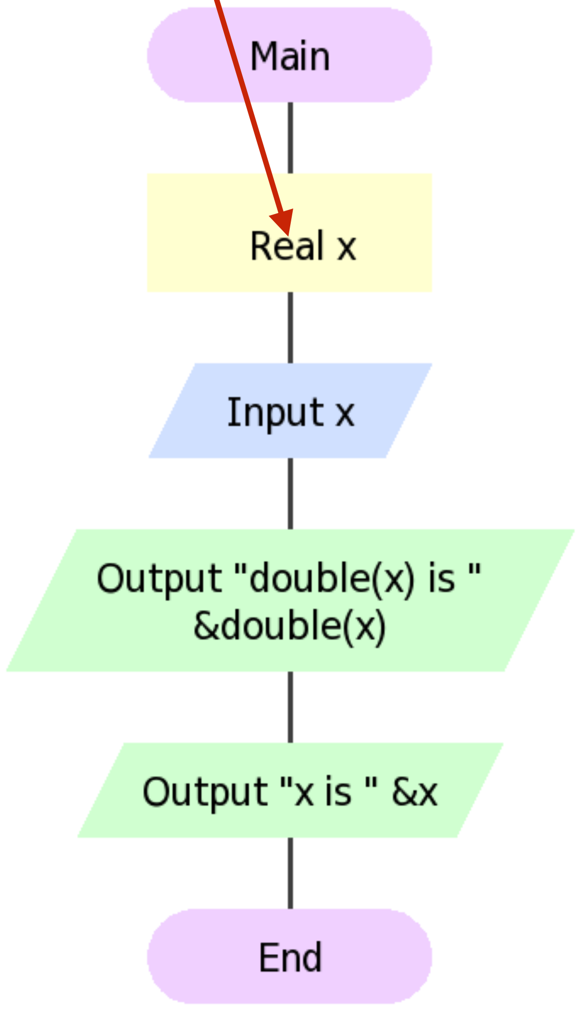
Scope

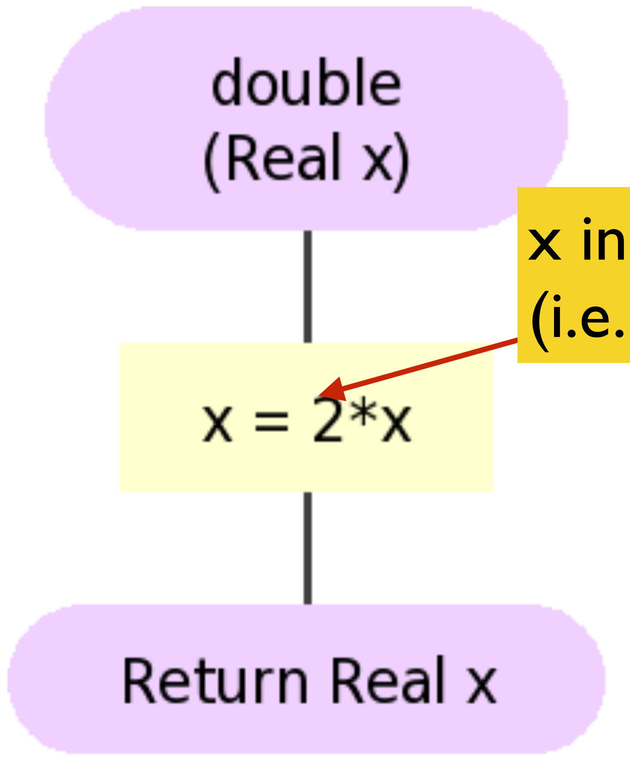
The scope of a function refers to where a variable can be used.

- In virtually all languages, the variables defined within a function only apply to that function. They have local scope.
- Similarly except in special cases, variables defined outside the function cannot be accessed within the function, i.e. they also have local scope.
- Variables that can be accessed everywhere, even inside functions, have global scope. Generally they're used only for constants. We will explain this later.

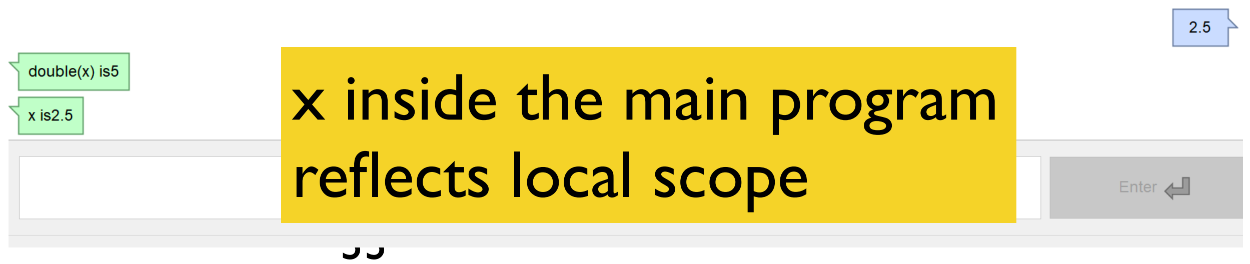
Scope in Flowgorithm

x has local scope





x inside the function is a different variable (i.e. copy)



```
double(x) is5
x is2.5
```

```
2.5
```

x inside the main program reflects local scope

4. Implementing functions in Python

Introduction

- We’ve already learned almost everything we need to know about functions.
- Implementation in Python is straightforward.

Defining a function in Python

A Python function is defined inside a `def` block:

arguments

```
def functionName ([args1,] […][argsn]):
```

name

```
    statement 1.1

    statement 1.2

    […]

     [return] ([expression])
```

Statements executed within the function are indented by the same amount

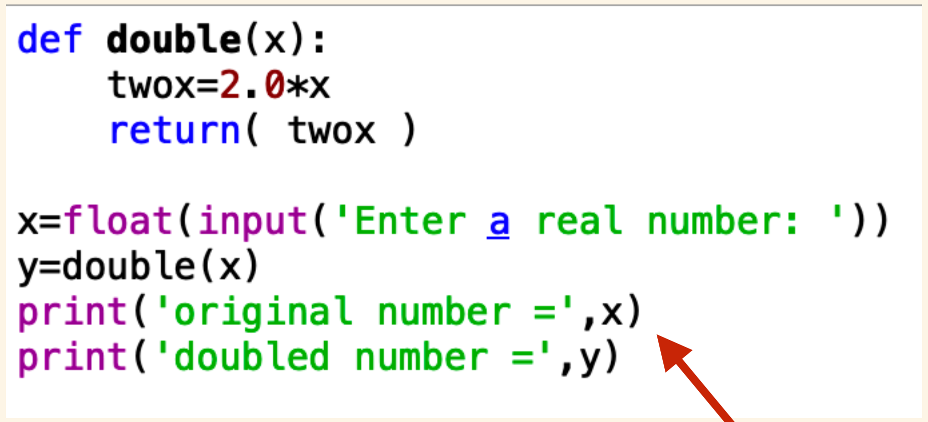
return value

Calling a Python function

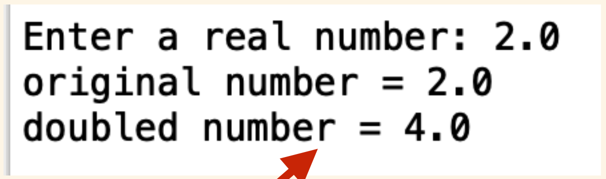
- A user-defined functions can be used in exactly the same way as any other function in Python (e.g. `len, str, abs`).
- For a function `func()`:

`z = func(x)`	z stores return value
`z = func(x,y)`	Multiple arguments
`func(x)`	No return value
`a,b = func(x)`	Multiple return values
`print(‘output=‘, func(x))`	Return value incorporated into expression

Example 1: doubling a real number



```
def double(x):
    twox=2.0*x
    return( twox )

x=float(input('Enter a real number: '))
y=double(x)
print('original number =',x)
print('doubled number =',y)
```

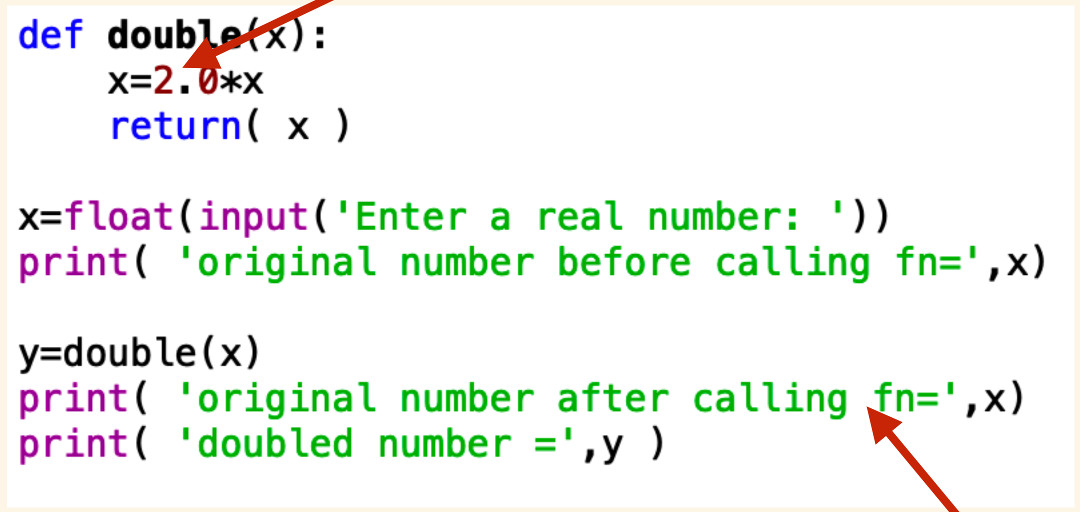


```
Enter a real number: 2.0
original number = 2.0
doubled number = 4.0
```

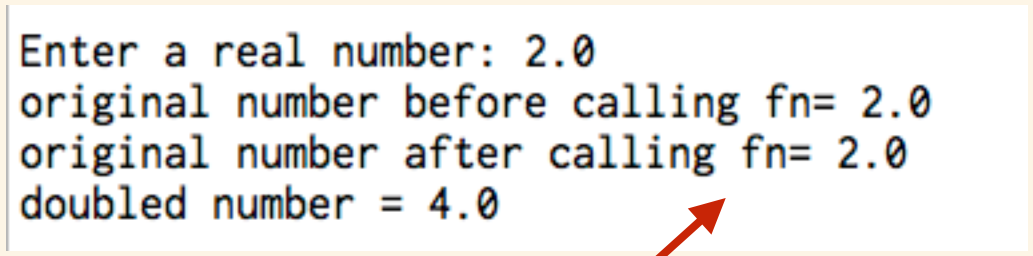
The original number is unchanged.

Example 2: using the same variable name in the function

x is modified inside the function



```
def double(x):
    x=2.0*x
    return( x )

x=float(input('Enter a real number: '))
print( 'original number before calling fn=',x)

y=double(x)
print( 'original number after calling fn=',x)
print( 'doubled number =',y )
```



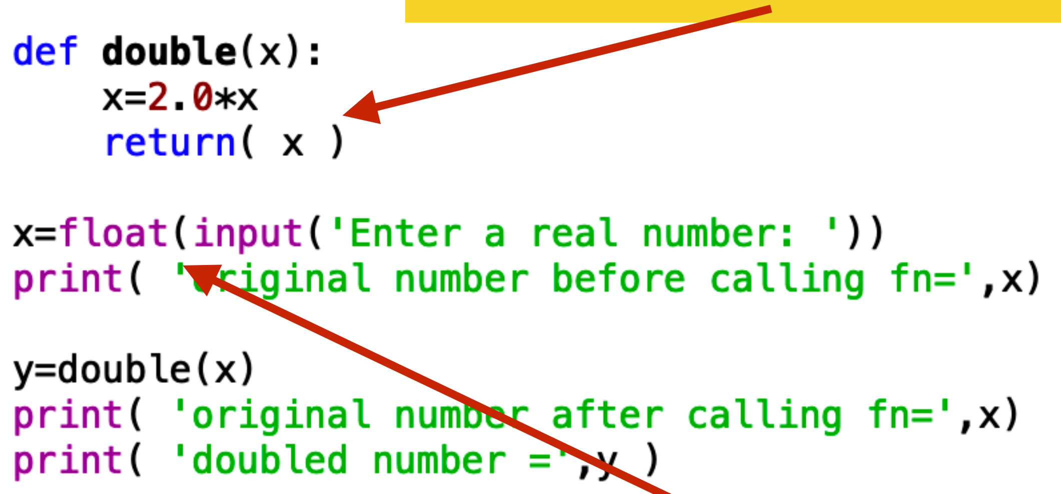
```
Enter a real number: 2.0
original number before calling fn= 2.0
original number after calling fn= 2.0
doubled number = 4.0
```

The original number is unchanged. Python is (essentially) pass by value.

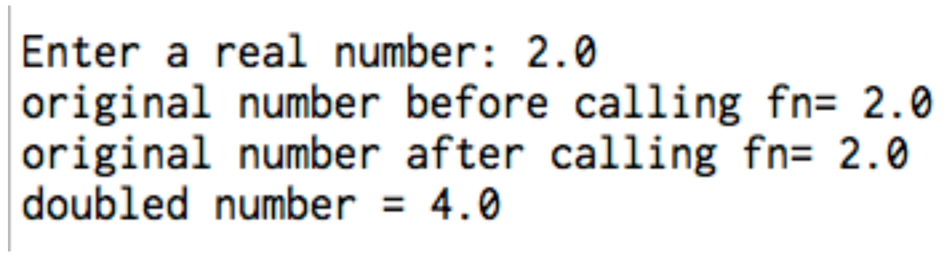
Local scope in Python

Modifications to a variable inside a Python function stay inside the function! This is local scope.

x is defined inside the function



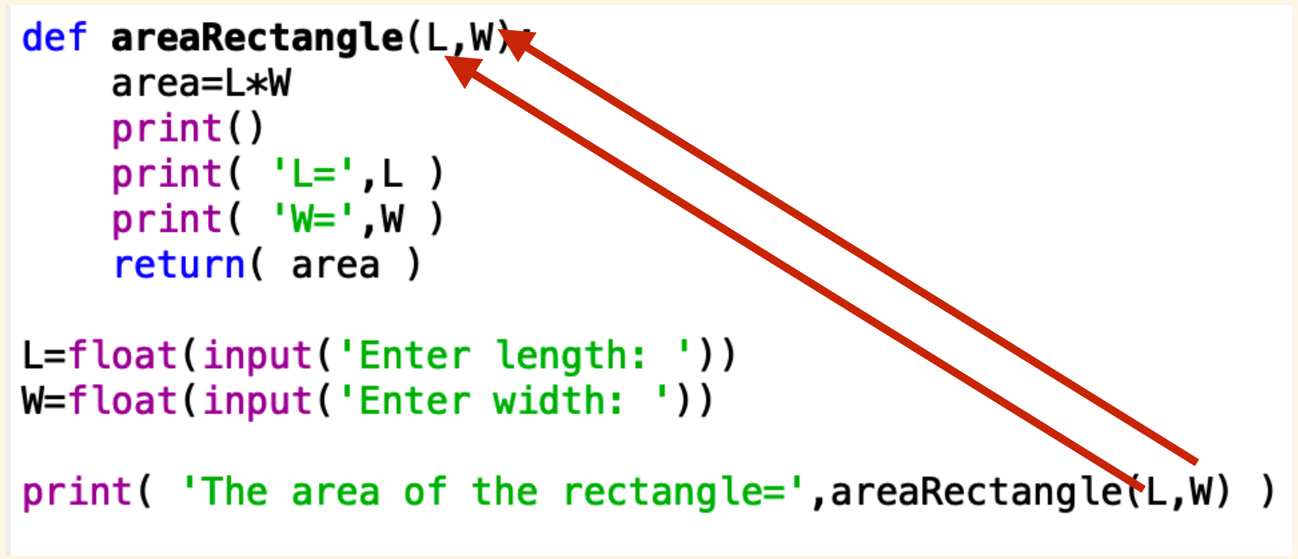
```
def double(x):
    x=2.0*x
    return( x )

x=float(input('Enter a real number: '))
print( 'original number before calling fn=',x)

y=double(x)
print( 'original number after calling fn=',x)
print( 'doubled number =',y )
```



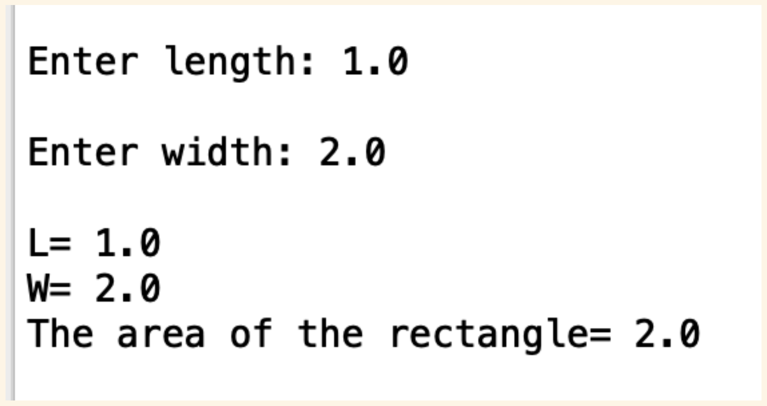
```
Enter a real number: 2.0
original number before calling fn= 2.0
original number after calling fn= 2.0
doubled number = 4.0
```

x is defined outside the function

Example 3: using multiple arguments



```
def areaRectangle(L,W):
    area=L*W
    print()
    print( 'L=',L )
    print( 'W=',W )
    return( area )

L=float(input('Enter length: '))
W=float(input('Enter width: '))

print( 'The area of the rectangle=',areaRectangle(L,W) )
```

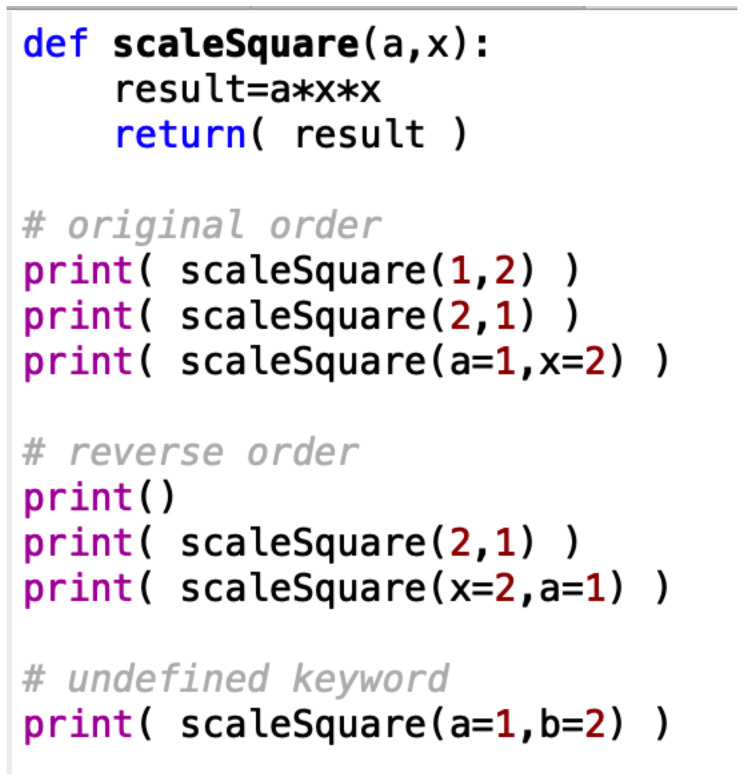


```
Enter length: 1.0

Enter width: 2.0

L= 1.0
W= 2.0
The area of the rectangle= 2.0
```

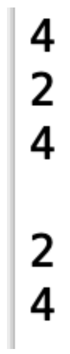
The order in which the argument is passed is crucial

Named arguments

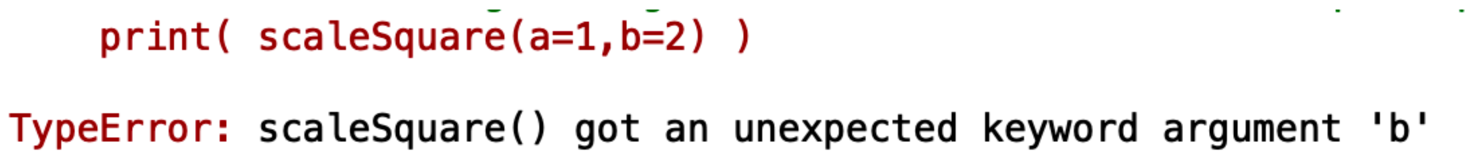
To avoid confusion, we can name the arguments. In this case the order of the arguments doesn't matter:



```python
def scaleSquare(a,x):
    result=a**x
    return( result )

# original order
print( scaleSquare(1,2) )
print( scaleSquare(2,1) )
print( scaleSquare(a=1,x=2) )

# reverse order
print()
print( scaleSquare(2,1) )
print( scaleSquare(x=2,a=1) )

# undefined keyword
print( scaleSquare(a=1,b=2) )
```



```
4
2
4

2
4
```

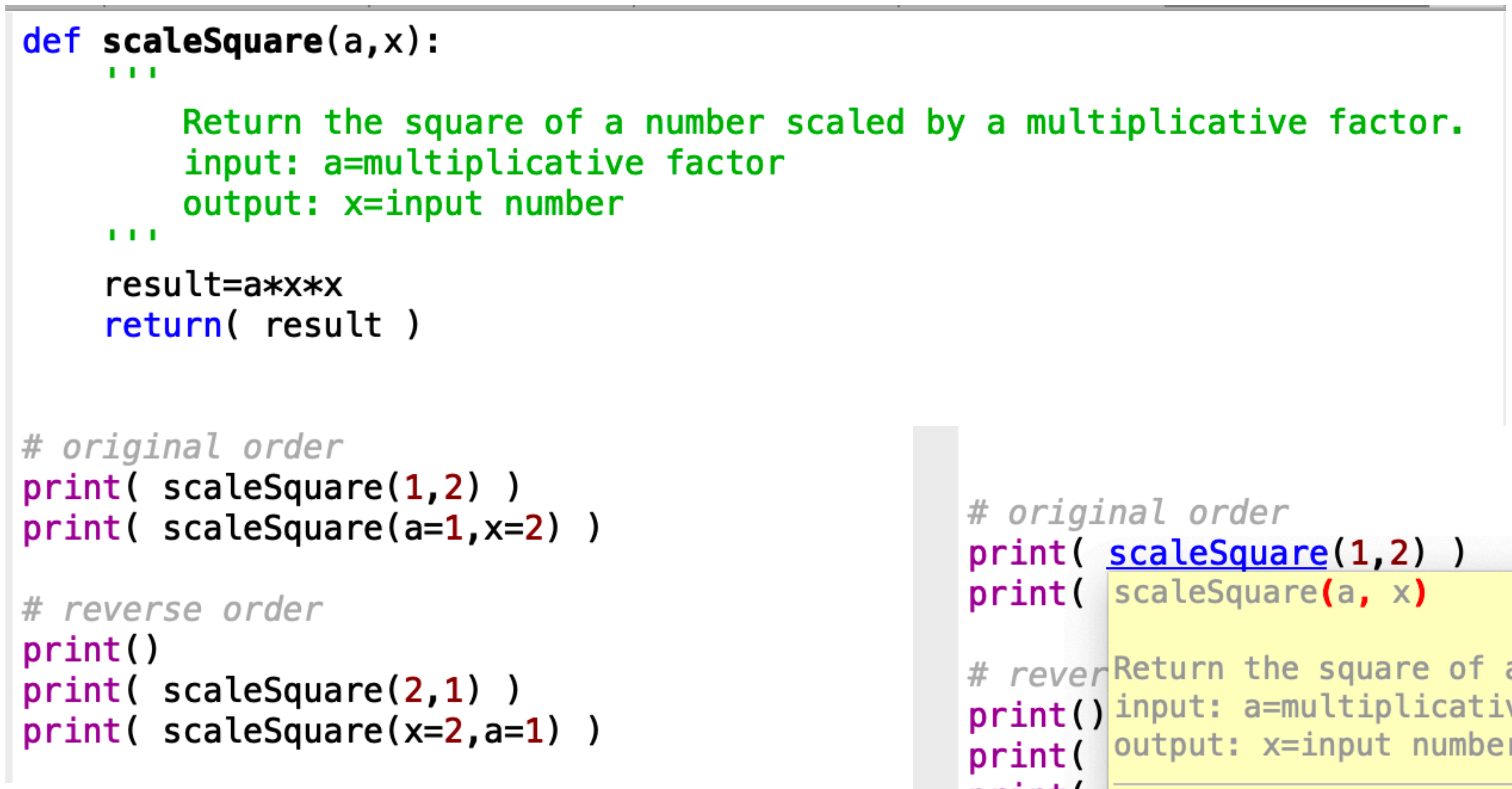


```
    print( scaleSquare(a=1,b=2) )

TypeError: scaleSquare() got an unexpected keyword argument 'b'
```

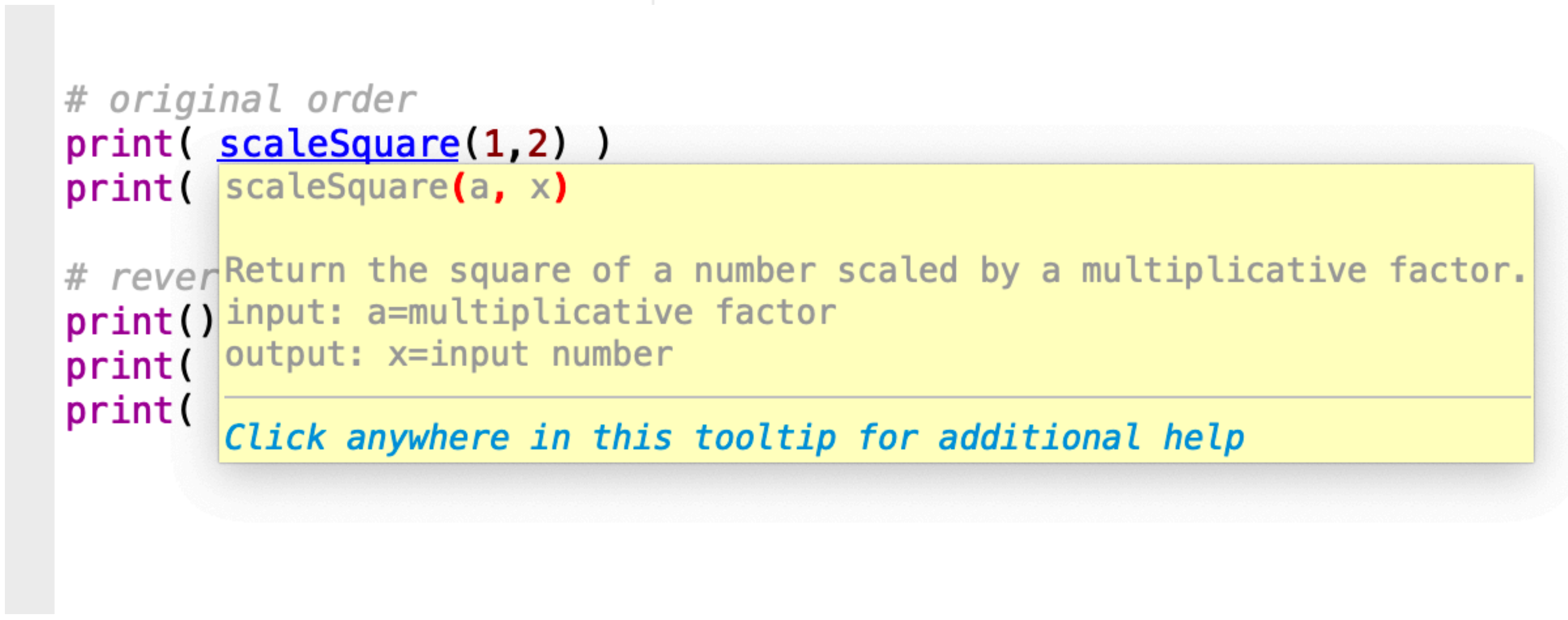
We will not use named arguments much because our functions are pretty simple. However, they can be useful for more complicated functions and programs.

Documenting functions

We can include information about our function inside a Python docstring defined by `'''` and `'''`. If the doctoring is defined, Spyder will show this information when we hover over the function name.



```
def scaleSquare(a,x):
    '''
        Return the square of a number scaled by a multiplicative factor.
        input: a=multiplicative factor
        output: x=input number
    '''
    result=a*x*x
    return( result )


# original order
print( scaleSquare(1,2) )
print( scaleSquare(a=1,x=2) )

# reverse order
print()
print( scaleSquare(2,1) )
print( scaleSquare(x=2,a=1) )
```



```
# original order
print( scaleSquare(1,2) )
print(
```

```
scaleSquare(a, x)

Return the square of a number scaled by a multiplicative factor.
input: a=multiplicative factor
output: x=input number

Click anywhere in this tooltip for additional help
```

```
# rever
print()
print(
print(
```

Comments

1. Arguments are optional.
2. The return value is optional.
3. `return` is optional, but it’s good practice to include it.
4. Variables defined inside a Python function have local scope.
5. For our purposes, variables passed to a Python function are passed by value (i.e. only a copy is passed to the function).

Summary

1. Functions are important for promoting code reuse.
2. The use of functions improves the structure of a program. In structured programming, each well-defined task should be assigned to a separate code block or function.
3. In computer programming, a function is defined by its name, argument(s) and return value(s).
4. In Python, a function need not include arguments or return values.