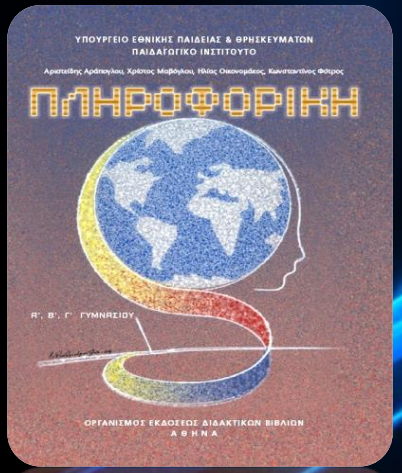
http://www.zioulas.gr

VCZ

LOGO PROCEDURES

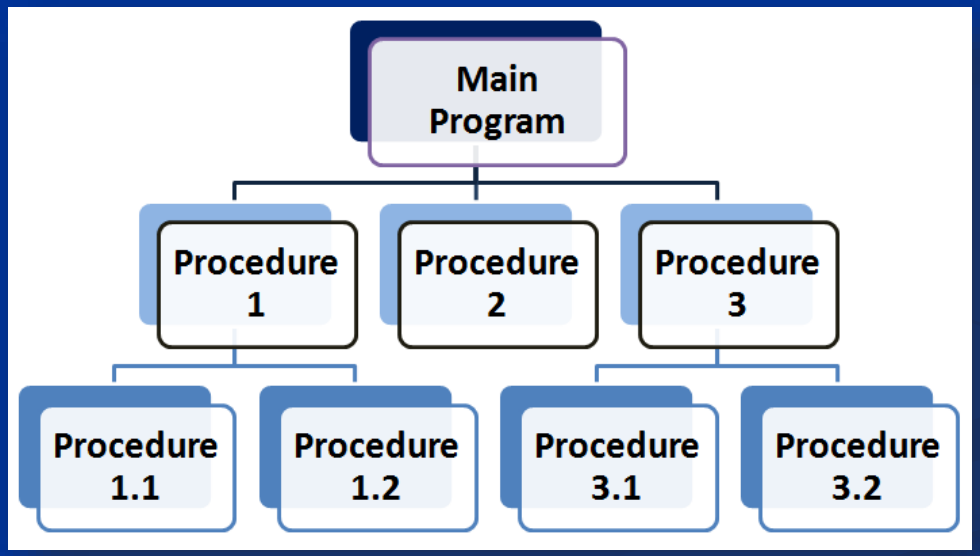
CHAPTER 7



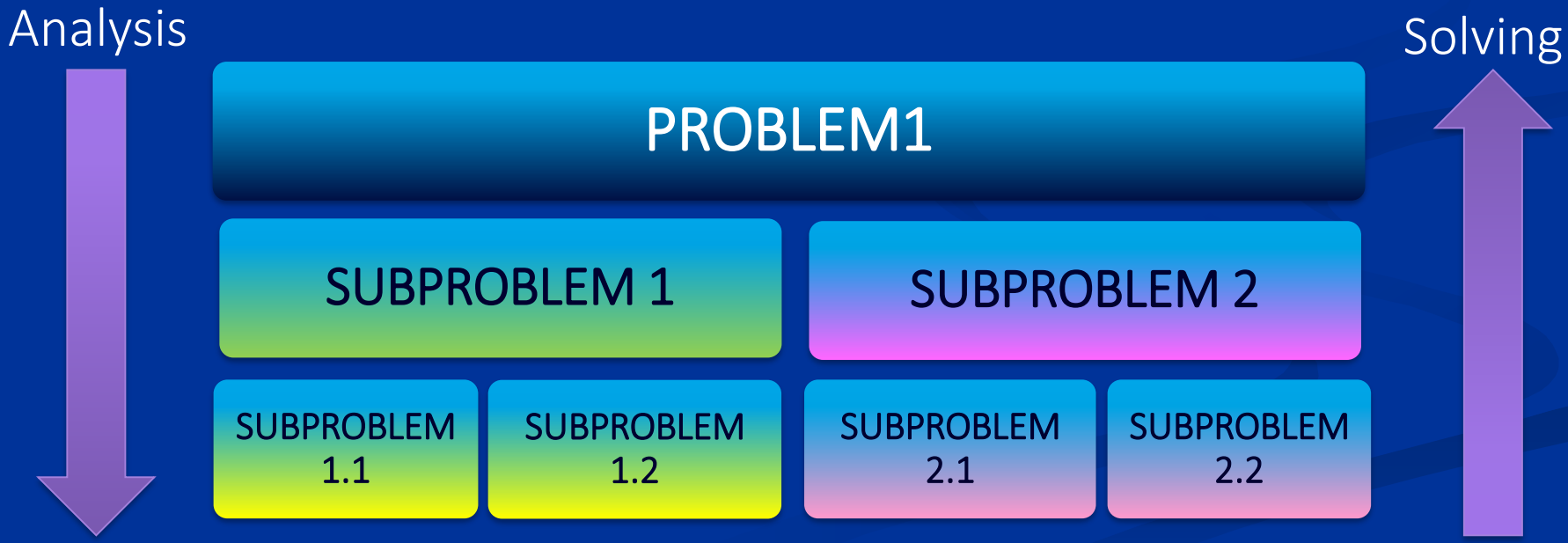
EVANGELOS C. ZIOULAS (IT TEACHER)

PROCEDURAL PROGRAMMING

- In Computer Science the **Procedural Programming** is a philosophy of creating programs as a set of individual sub-programs.

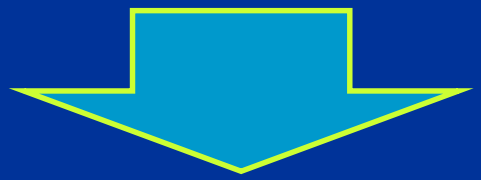


- This programming philosophy is based on the **divide and conquer** principle, since it breaks the initial problem into individual **sub-problems** (also known as **tasks**).
- Each complicated task is divided into individual sub-tasks, until the final tasks are to be fully comprehensible and easily resolved by the programmer.



PROCEDURE

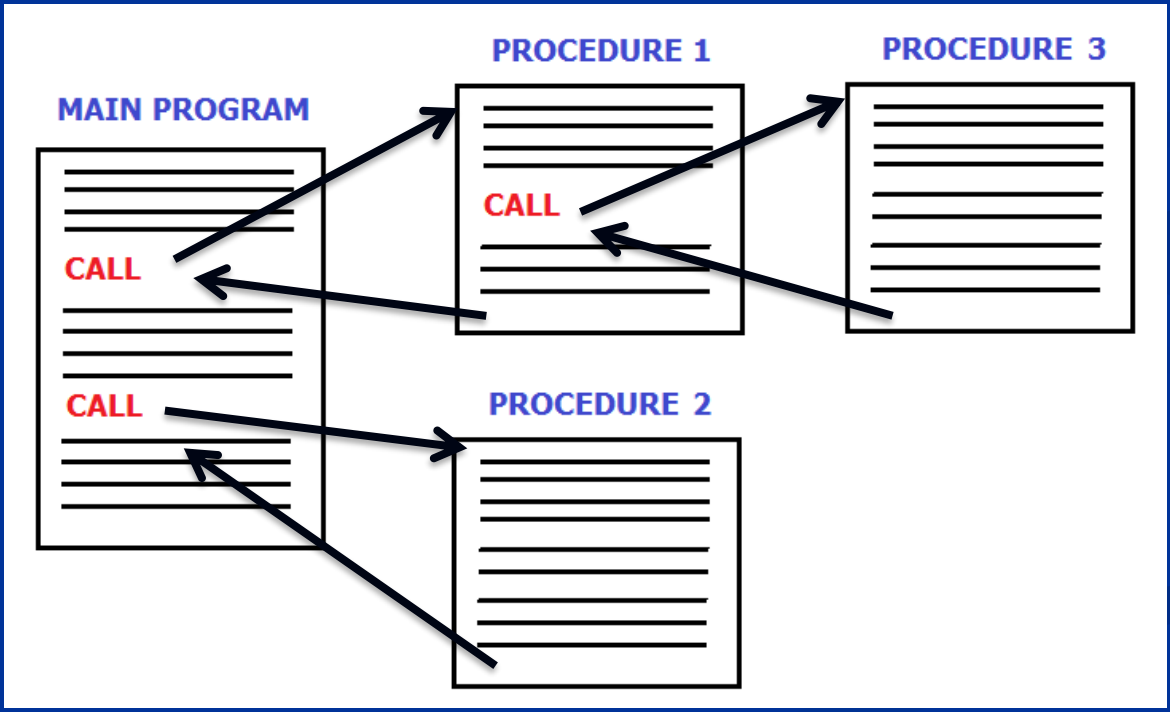
A **procedure** which is also known as **routine**, **subroutine**, **method**, or **function**, is a set of computational steps that needs to be executed to resolve a particular task.



A **procedure** is a list of instructions with has a name.
Once we have created a procedure, we just type its name to run all the instructions it contains.
Its instructions are executed sequentially one after the other.

THE ADVANTAGE OF A PROCEDURE

- The main advantage of a procedure is that **we can call it**, when needed, at any point of the program execution by **using only its name** without having to write again its set of commands.



PUBLIC PROCEDURES

- To create a **public procedure** that is known to all the turtles, we should define it inside the **Procedures** tab.

- All procedures must start with **to** and the **name** of the procedure (title line).
- Procedures must also end with the word **end** that must be on its own line (last line).

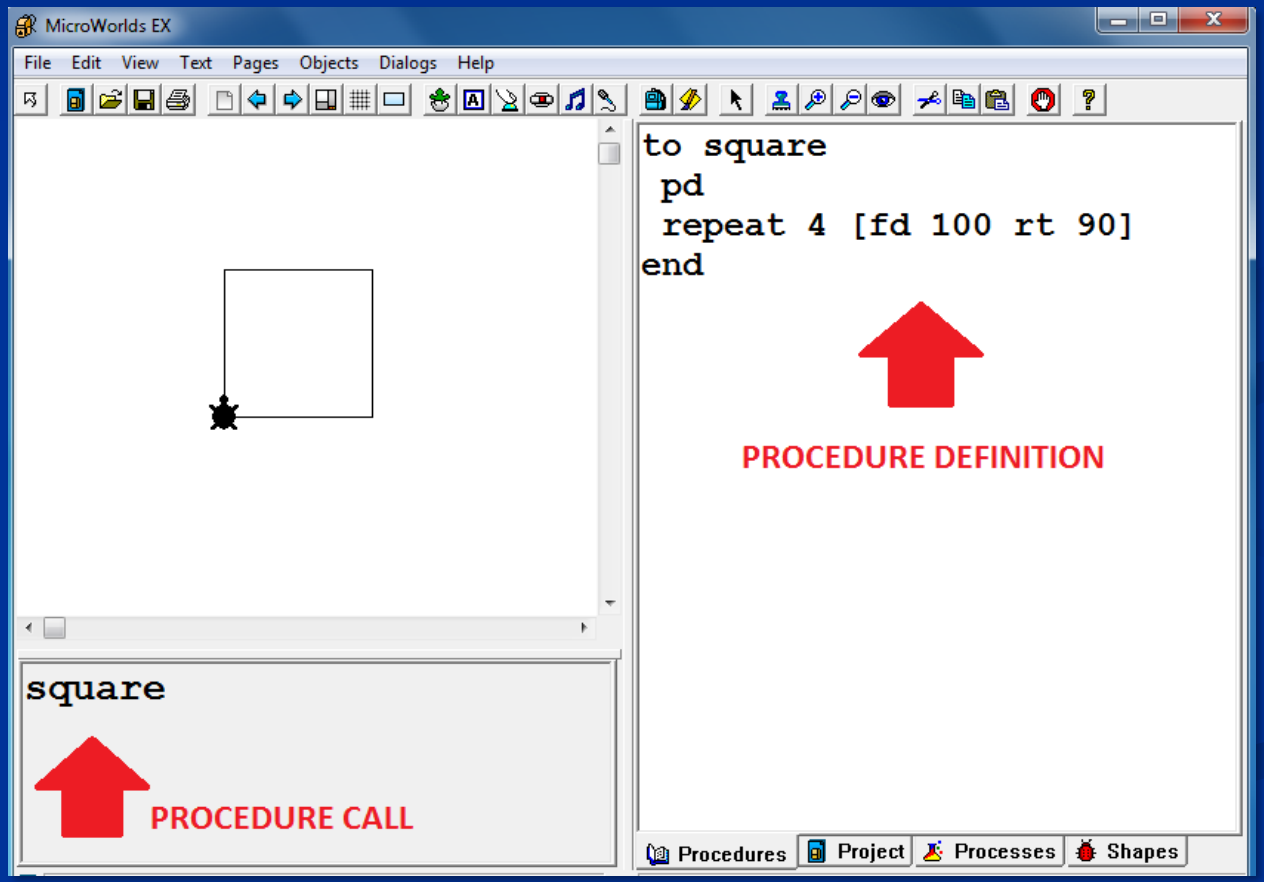
DEFINITION OF A PUBLIC PROCEDURE

```
to procedure_name
        command 1
        command 2
        ........
end
```

We can choose any name we want, but always we should make sure that it doesn't contain any spaces.

PROCEDURE CALL

To call a procedure, we should type its name in the **Command Center** or use it as part of an instruction in a Turtle's backpack or in a button.



IMPORTANT NOTES

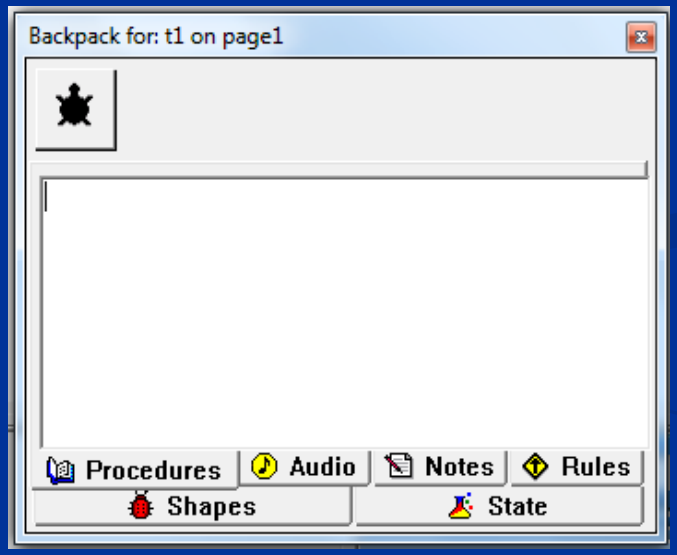
- If a procedure in a Turtle's backpack (a **Private procedure**) has the same name as a procedure in the project Procedures tab (a **Public procedure**), that specific Turtle uses the Private procedure, never the Public one.

- If we call in the Command Center a procedure that doesn't exist, MicroWorlds cannot find its definition and give us back an error message: **I don't know how to …**

- When we define a procedure, it becomes **part of MicroWorlds' vocabulary for that project**. When we open another project or start a new project, this procedure is no longer available. Of course, we can copy and paste its definition into the new project.

PRIVATE PROCEDURES

- The Procedures tab in a Turtle's backpack is where we define **Private procedures**. Only the Turtle with the procedure in its backpack "knows" this procedure and can run it.



To display the backpack's Procedures tab, we should open the Turtle's backpack and click on the Procedures tab.

START-UP PROCEDURE

- If we need something to happen immediately as we open a project, we should put the appropriate instructions inside a procedure named **startup**, in the Public procedures tab.

```
to startup
  announce [Welcome to my World!]
  announce [This is MicroWorlds EX]
  page1
end
```

- The startup procedure is also a good way to set up variables and other objects in a project.
- The startup procedure acts like a regular procedure and cannot take any parameter as an input.

```
to startup
  t1, setc “blue
  t2, setsize 100
  t2, setpensize 30
  setbg 55
end
```

PROCEDURE TYPES

- In MicroWorlds EX programming environment we can create 3 types of procedures:

  A) Regular procedures

  B) Parametric procedures

  C) Super procedures

REGULAR PROCEDURES

- They are procedures that do not use any parameter or variable.

```
to square
 repeat 4 [fd 100 rt 90]
end
```

call: square

```
to rectangular
 repeat 2 [fd 100 rt 90 fd 50 rt 90]
end
```

call: rectangular

PARAMETRIC PROCEDURES

- In a regular procedure, we draw exactly the same shape each time it runs.
- The parametric are procedures that use a list of **parameters** (**variables**) as input.
- Since a procedure takes a particular input, we can use it to draw shapes of all different lengths.

- The square procedure can be modified so that instead of drawing a square with a side of 100 each time, it can draw squares of all different sizes.
- Each time that we want to run the procedure **square**, we must specify the length of the side of the square to be drawn.

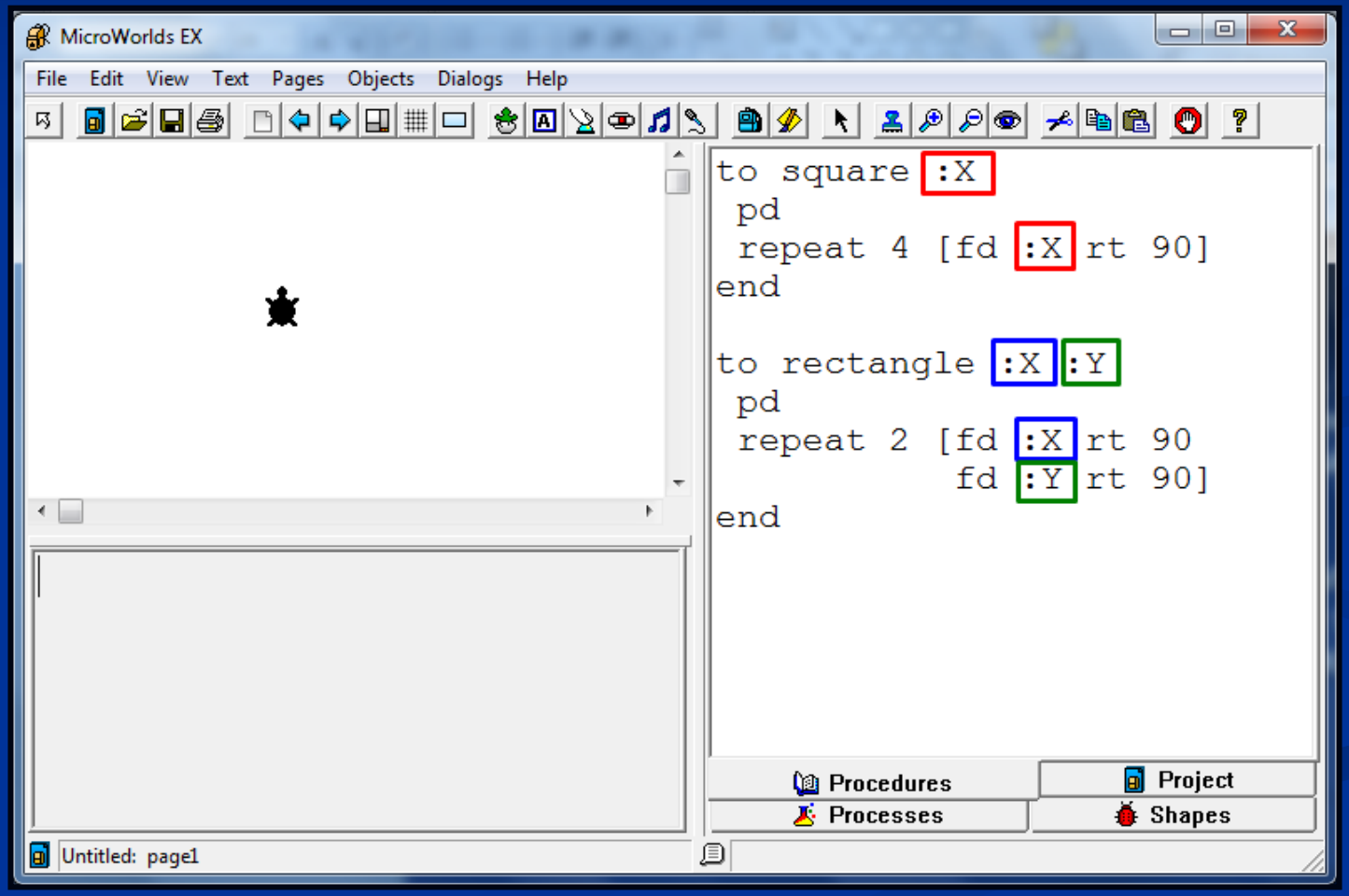
```
to square :X
 repeat 4 [fd :X rt 90]
end
```

call: square 50

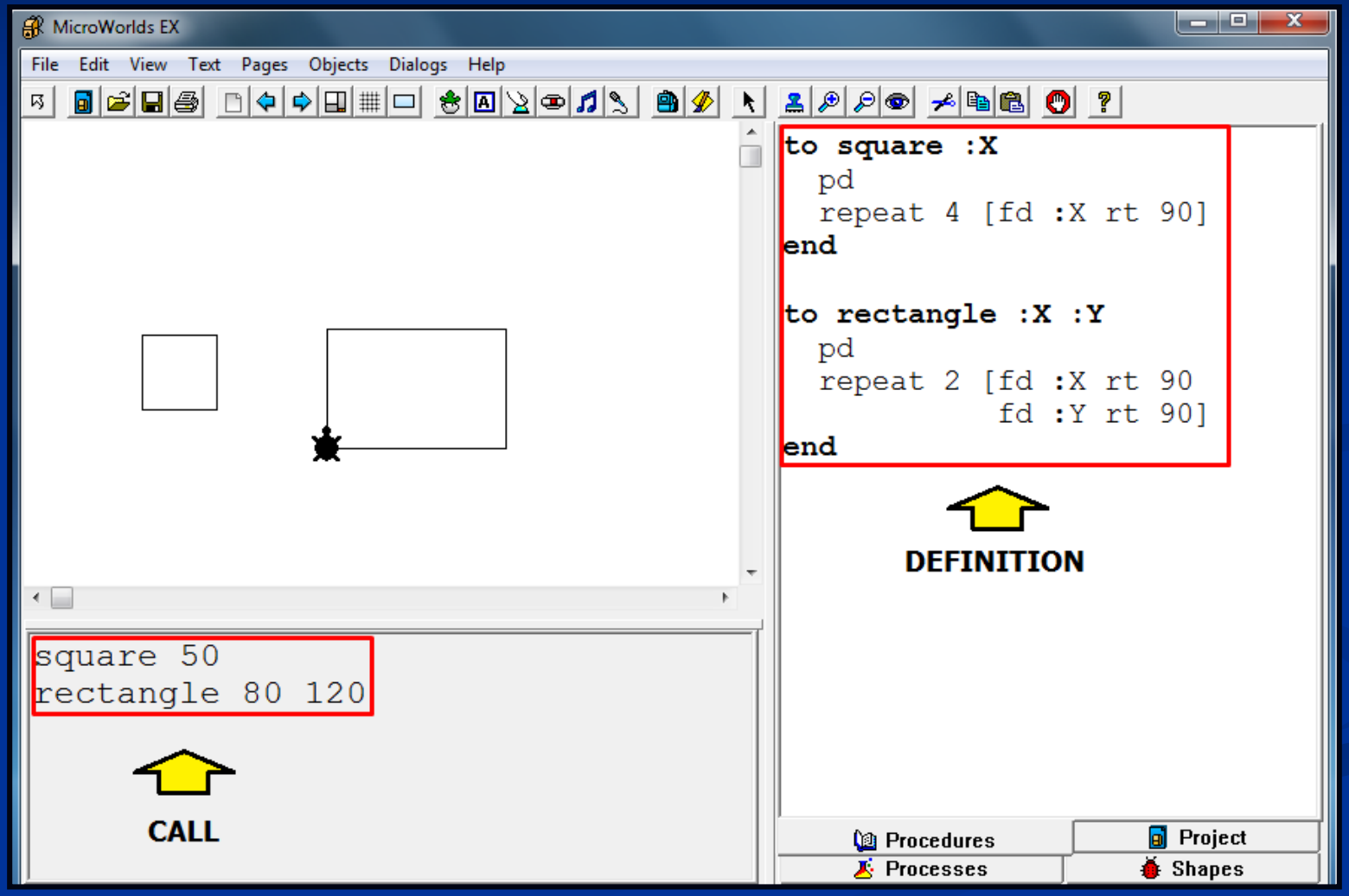
```
to rectangular :X :Y
 repeat 2 [fd :X rt 90 fd :Y rt 90]
end
```

call: rectangular 200 100

- The name of the input must be written on the title line after the name of the procedure. We can choose any name for the input, but it must always be **preceded by a colon** (**:**) e.g. :X,:Y

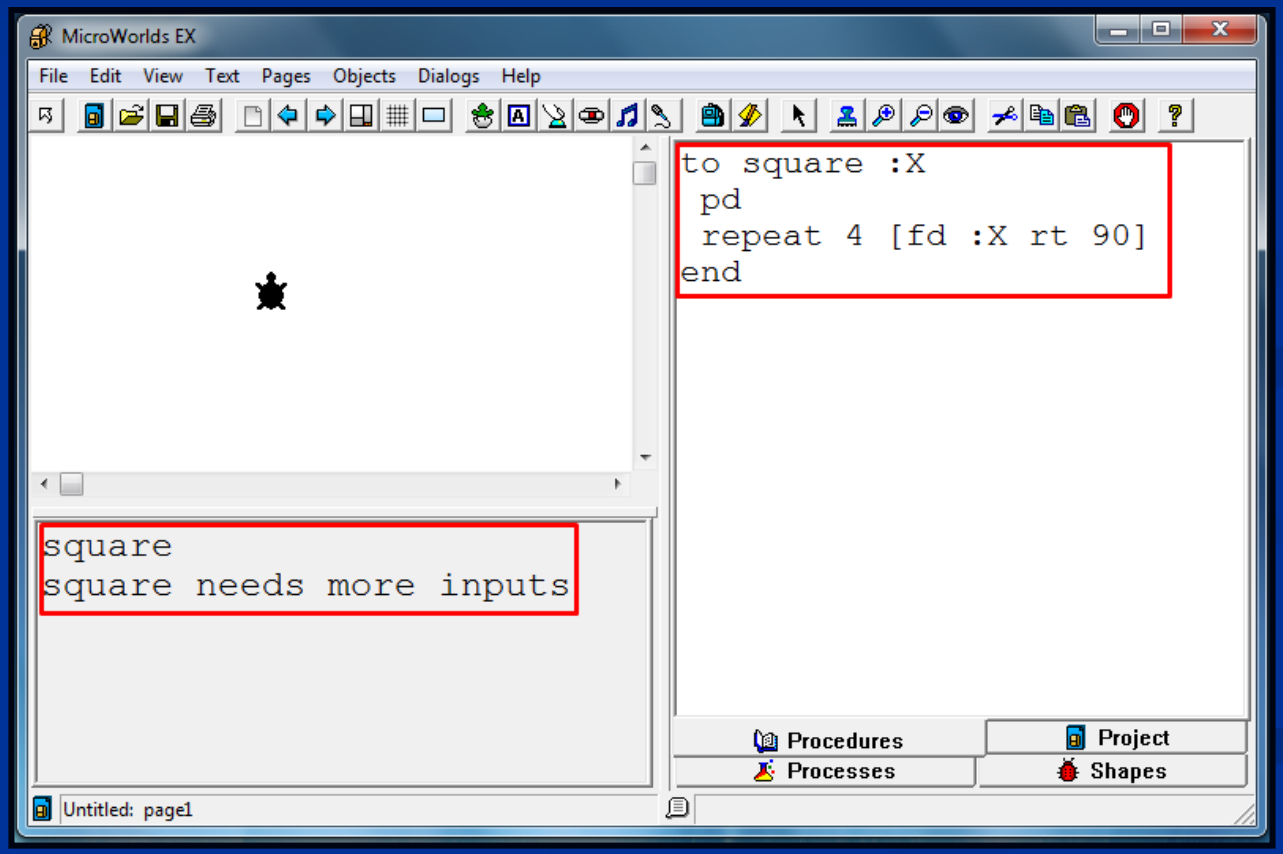


We can use the input name (preceded by the colon) wherever we want to use the value of that input in the procedure.



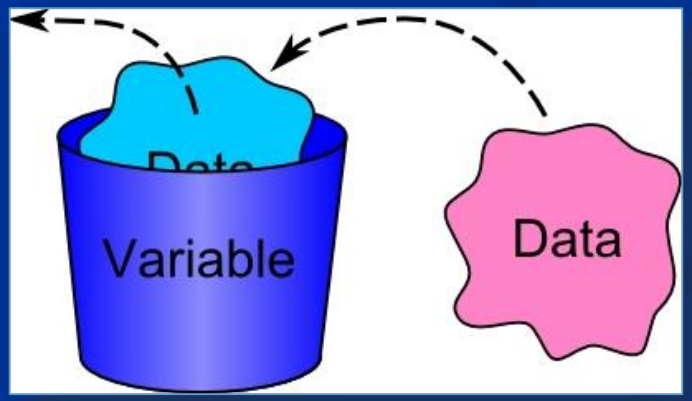
EXAMPLE OF A WRONG CALL

- If MicroWorlds expects an input for a procedure, it complains if we do not include one:



VARIABLES

- A variable is a **symbolic name** which is given to a location of computer memory.
- In this location we can store temporarily a value of a program.
- The value of a variable can be changed during the program execution, which means that its previous value is lost and cannot be used again.



VARIABLES

- A variable is a memory space that is able to store only one value per time.
- **To access the value** of a variable we should place in front of its name the **: symbol.**
- **To assign a new value** to a variable, we place in front of its name the **" symbol.**

THE MAKE COMMAND

ASSIGNMENT COMMAND

- It **creates a variable** and **gives it a** particular **value**.
- When we assign a value to a variable this value might be a number, a word or a list.
- Variables created by **make** command keep their values as long as we don't clear them or quit MicroWorlds EX. However, they are not saved with our project.

```
make "A 5
show :A
5
clearname "A
show :A
A has no value
```

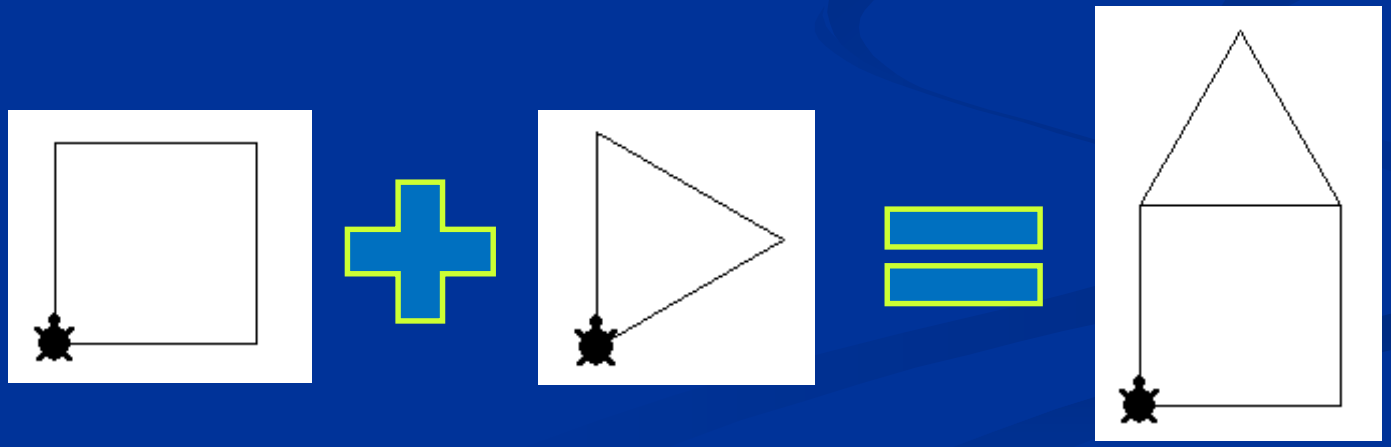
EXAMPLES OF VARIABLES

`make "X 25`	It assigns the value 25 to the variable X
`show :X` `25`	It displays the value of the variable X
`show sqrt :X` `5`	It displays the square root of the variable X
`make "Y "Hello`	It assigns the value “Hello to the variable Y
`show :Y` `Hello`	It displays the value of the variable Y
`show (se :Y "students)` `Hello students`	It displays the sentence *Hello students*

SUPER PROCEDURES

- They are procedures that **include** (call) **other procedures** into their instructions.

- If we create a square procedure and a triangle procedure then we are able to create the house procedure by calling the first two sub-procedures:

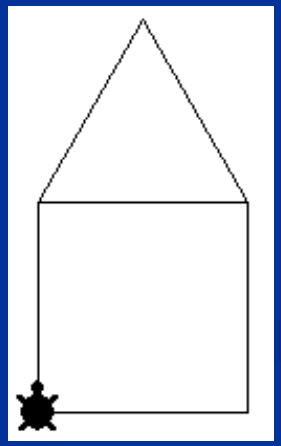


EXAMPLE

```
to square
 repeat 4 [fd 100 rt 90]
end
```

```
to triangle
 repeat 3 [fd 100 rt 120]
end
```

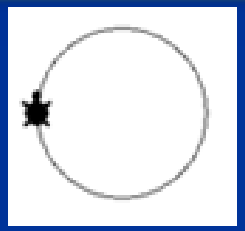
```
to house
 pd
 square
 fd 100 rt 30
 triangle
 lt 30 bk 100
end
```



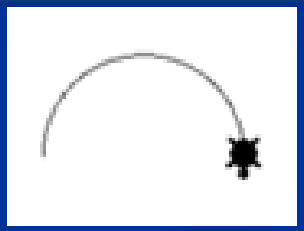
call: house

EXAMPLE

```
to circle
  repeat 360 [fd 1 rt 1]
end
```



```
to semicircle
  repeat 180 [fd 1 rt 1]
end
```

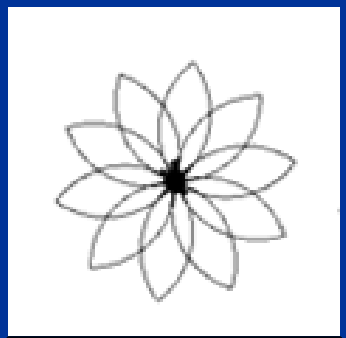


```
to quadrant
repeat 90 [fd 1 rt 1]
end
```

```
to petal
  repeat 2 [quadrant rt 90]
end
```

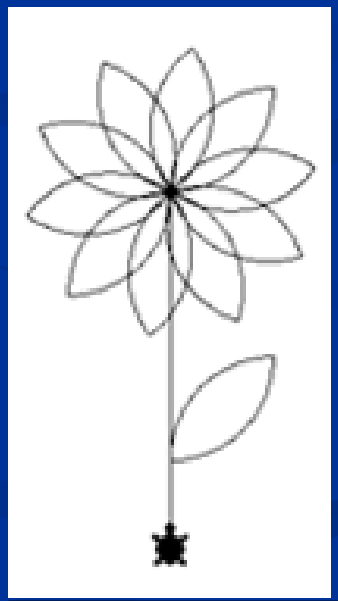
EXAMPLE

```
to flower
 repeat 10 [petal rt 36]
end
```



call: flower

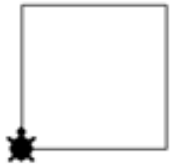
```
to plant
 flower
 bk 150 petal bk 50
end
```



call: plant

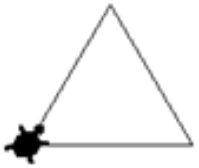
EXAMPLES OF PARAMETRIC SUPER PROCEDURES

```
to square :size
repeat 4 [fd :size rt 90]
end
```



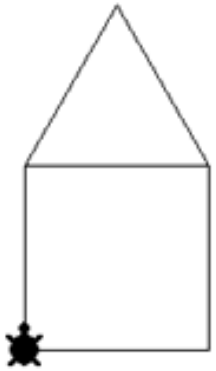
call: **square 150**

```
to triangle :size
repeat 3 [fd :size rt 120]
end
```



call: **triangle 150**

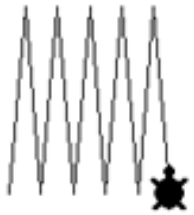
```
to house :size
 square :size
 fd :size
 triangle :size
 lt 30
 bk :size
end
```



call: **triangle 150**

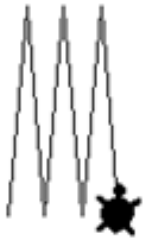
EXAMPLES OF PARAMETRIC SUPER PROCEDURES

```
to zigzag :size
 rt 5
 repeat 5
  [fd :size rt 170
   fd :size lt 170]
end
```



call: **zigzag 100**

```
to zigzag :num :size :angle
 rt 5
 repeat :num
  [fd :size rt :angle
   fd :size lt :angle]
end
```



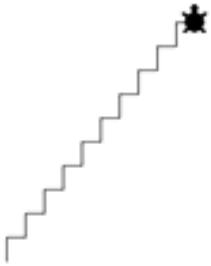
call: **zigzag 3 100 170**

EXAMPLES OF PARAMETRIC SUPER PROCEDURES

```
to staircase :steps
 repeat :steps
   [fd 30 rt 90
    fd 20 lt 90]
end
```

call: **staircase 5**

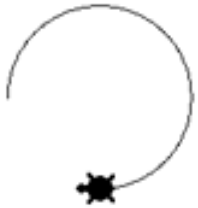
```
to staircase :steps :h :w
 repeat :steps
   [fd :H rt 90
    fd :w lt 90]
end
```



call: **staircase 10 20 15**

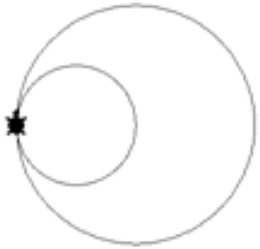
EXAMPLES OF PARAMETRIC SUPER PROCEDURES

```
to arc :size
 repeat :size [fd 1 rt 1]
end
```



call: **arc 270**

```
to circle :X
 repeat 360 [fd :X rt 1]
end
```



call: **circle 1**
call: **circle 2**

```
to quadrant :X
 repeat 90 [fd :X rt 1]
end
```

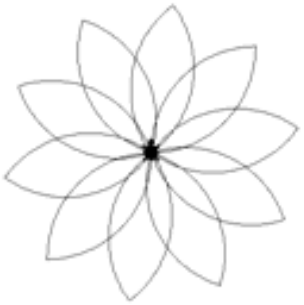
call: **quadrant 2**

EXAMPLES OF PARAMETRIC SUPER PROCEDURES

```
to petal :X
 repeat 2 [quadrant :X
           rt 90]
end
```

call: **petal 1**
call: **petal 2**
call: **petal 3**

```
to flower :X
 repeat 10 [petal :X
            rt 36]
end
```



call: **flower 2**

EXAMPLES OF PARAMETRIC SUPER PROCEDURES

```
to flower :num :size
 repeat :num
    [petal :size
     rt 360 / :petals]
end
```

call: **flower 6 2**

```
to plant :num :size
 flower :num :size
 bk 50
 petal :size
 bk 50
end
```

call: **plant 15 1**