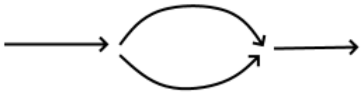
What is a procedure

July 27, 2022

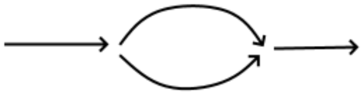
Control flow (continued)

- Using conditionals we can split the path of a program into two branches



Control flow (continued)

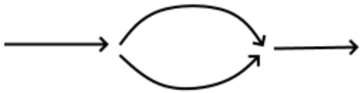
- Using conditionals we can split the path of a program into two branches



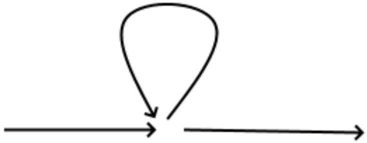
- Sometimes we want to repeat an action

Control flow (continued)

- Using conditionals we can split the path of a program into two branches

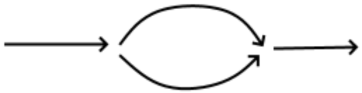


- Sometimes we want to repeat an action
- This is realized by *loops*

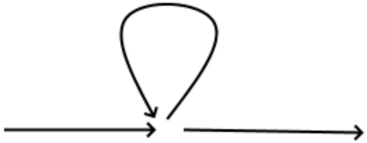


Control flow (continued)

- Using conditionals we can split the path of a program into two branches

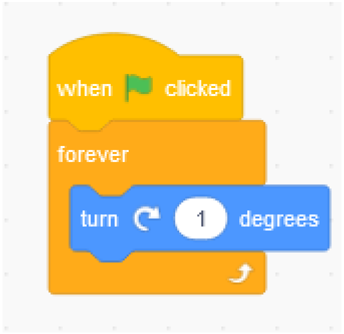


- Sometimes we want to repeat an action
- This is realized by *loops*



- For example, ask for a password until correct, retry connecting 3 times, ...

Spin


```
when [flag] clicked
forever
    turn ↻ (1) degrees
```

Procedures

- Other times, we want to reuse code in different places

Procedures

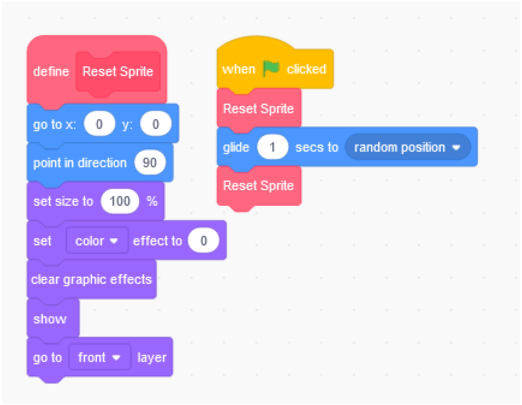
- Other times, we want to reuse code in different places
    - This doesn't always fit in a loop pattern. . .

Procedures

- Other times, we want to reuse code in different places
  - This doesn't always fit in a loop pattern...
- $\Rightarrow$ we use *procedures*

Procedures

- A procedure is simply a named block of code
  - which can be executed at any point after it is defined



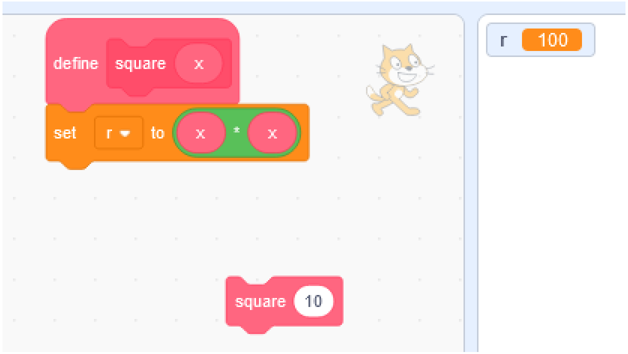
```
define Reset Sprite
go to x: 0 y: 0
point in direction 90
set size to 100 %
set color ▾ effect to 0
clear graphic effects
show
go to front ▾ layer
```

```
when 🏴 clicked
Reset Sprite
glide 1 secs to random position ▾
Reset Sprite
```

- (arbitrary large code block to illustrate that you really do not want to copy this two or more times)

Parameters

- Just like commands, procedures can take parameters

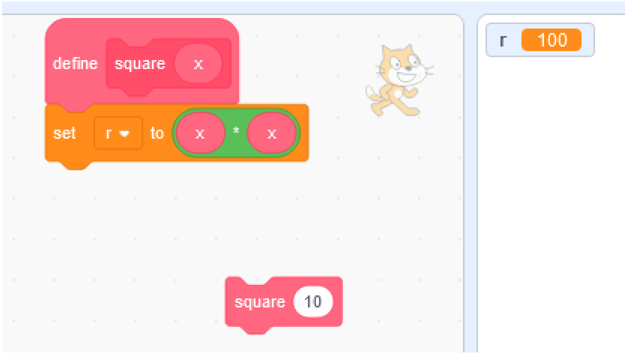


```
define square (x)
set r to (x * x)

square (10)
```

Parameters

- Just like commands, procedures can take parameters



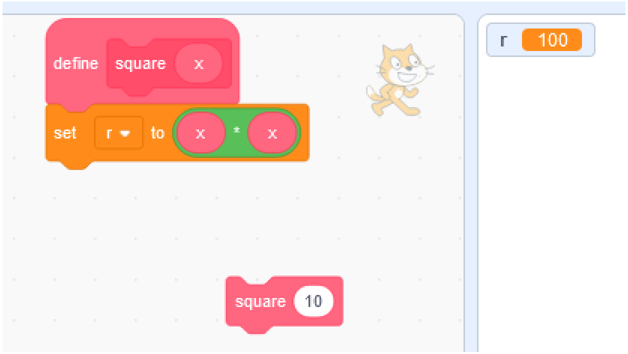
```
define square (x)
set r to ((x) * (x))

square (10)
```

- The “define” block defines a new block called “square”

Parameters

- Just like commands, procedures can take parameters



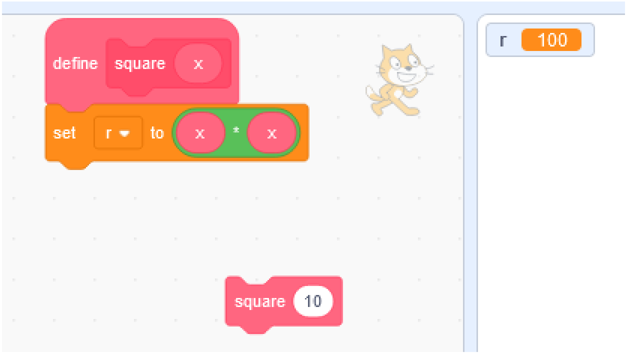
```
define square (x)
set r to (x * x)

square (10)
```

- The "define" block defines a new block called "square"
- When `square` is given a number and executed, it multiplies the number by itself, and writes it to `r`

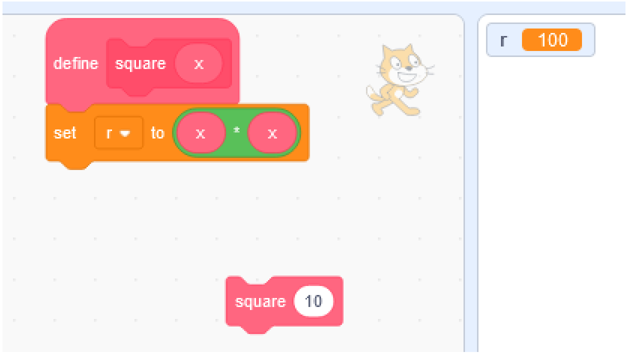
Parameters

- Just like commands, procedures can take parameters



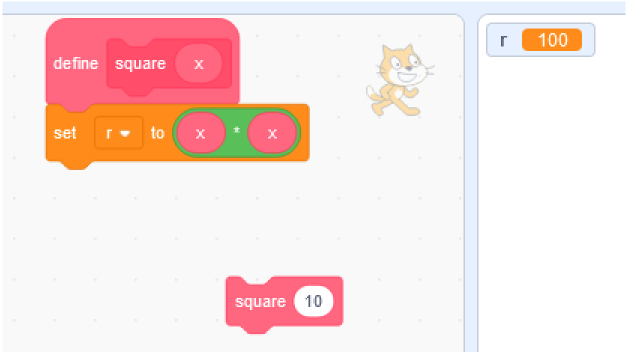
- The "define" block defines a new block called "square"
- When `square` is given a number and executed, it multiplies the number by itself, and writes it to `r`
    - E.g., running `square(10)` results in `r = 100`, running `square(-4)` would result in `r = 16`

Parameters



- In scratch, all variables are *global*, so the `r` inside `square` and outside are the same

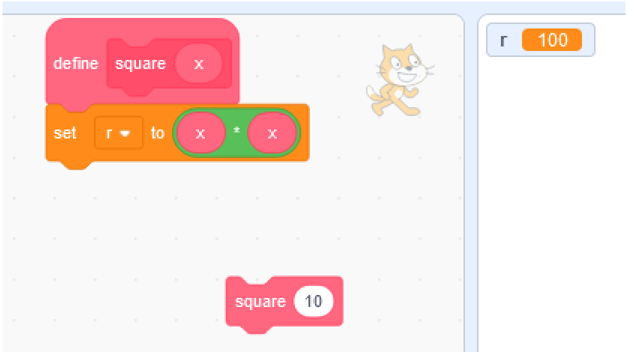
Parameters


```
define square (x)
set r to ((x) * (x))

square (10)
```

- In scratch, all variables are *global*, so the `r` inside `square` and outside are the same
  - Changing `r` also changes it on the outside

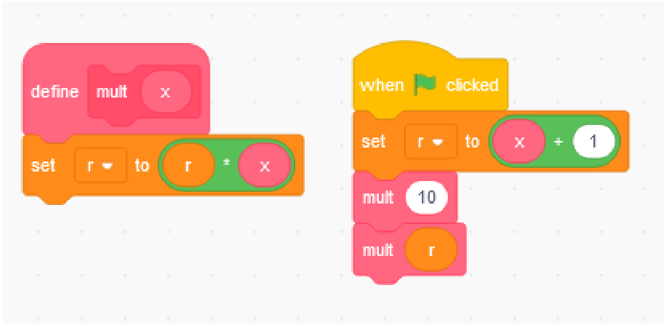
Parameters



- In scratch, all variables are *global*, so the `r` inside `square` and outside are the same
  - Changing `r` also changes it on the outside
- $\Rightarrow$ Procedures can introduce unexpected situations

Scoping

- What does this program do?

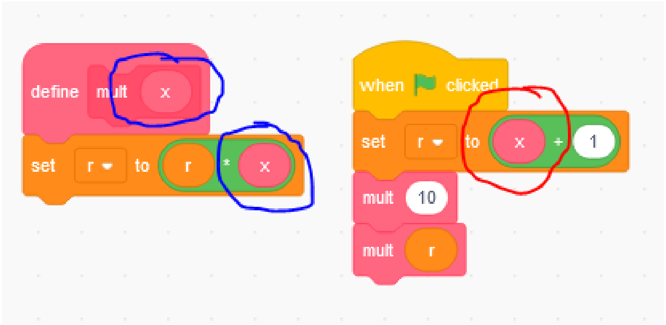


```
define mult (x)
set r to ((r) * (x))

when flag clicked
set r to ((x) + (1))
mult (10)
mult (r)
```

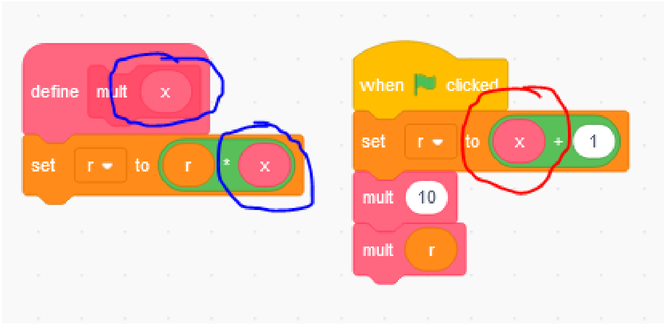
Scoping

- The `x` parameter is only defined inside `mult!` (⇒ it is *local* as opposed to global)
  - If it is used outside, Scratch always gives us 0 back



Scoping

- The `x` parameter is only defined inside `mult`! ($\Rightarrow$ it is *local* as opposed to global)
  - If it is used outside, Scratch always gives us 0 back



```
define mult (x)
set r to (r * x)
```

```
when 🏴 clicked
set r to (x + 1)
mult 10
mult r
```

- We say that `x` is *in scope* at the blue circles, while it is *out of scope* at the red circle

No new operators

- The behaviour of Scratch makes some things more simple

No new operators

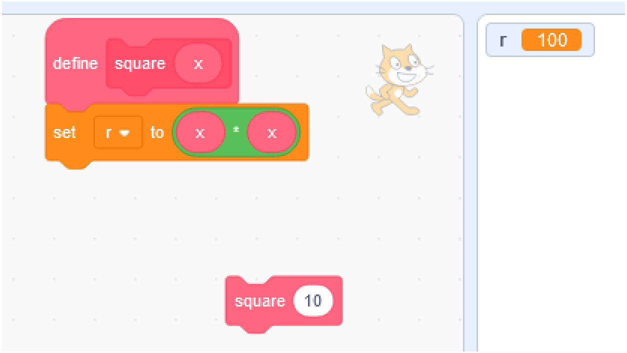
- The behaviour of Scratch makes some things more simple
- However, defining new operators is not possible

No new operators

- The behaviour of Scratch makes some things more simple
- However, defining new operators is not possible
  - This makes writing functions that compute a value a bit awkward

Square

- Recall



```
define square (x)
set r to ((x) * (x))

square (10)
```

- for which we had to create `r` to output the value of `x * x` to

More scoping is better

- The absence of local variables (excluding parameters) makes writing complicated functions even more complicated

More scoping is better

- The absence of local variables (excluding parameters) makes writing complicated functions even more complicated
  - Every intermediate value stored is globally visible

More scoping is better

- The absence of local variables (excluding parameters) makes writing complicated functions even more complicated
  - Every intermediate value stored is globally visible
- The ability to alter global variables at any point makes maintaining a large project more confusing

More scoping is better

- The absence of local variables (excluding parameters) makes writing complicated functions even more complicated
  - Every intermediate value stored is globally visible
- The ability to alter global variables at any point makes maintaining a large project more confusing
  - Every procedure can change any variable

More scoping is better

- The absence of local variables (excluding parameters) makes writing complicated functions even more complicated
  - Every intermediate value stored is globally visible
- The ability to alter global variables at any point makes maintaining a large project more confusing
  - Every procedure can change any variable
- Other languages offer alternatives

More scoping is better

- The absence of local variables (excluding parameters) makes writing complicated functions even more complicated
  - Every intermediate value stored is globally visible
- The ability to alter global variables at any point makes maintaining a large project more confusing
  - Every procedure can change any variable
- Other languages offer alternatives $\Rightarrow$ Python

Python vs Scratch

	Scratch	Python
Mode	Click and drag	Syntax
Procedures	Procedures	Functions
Variables	Global	Mixed
Scoping	No	Yes
Types	4	Many and more

What does this mean?

- Instead of clicking and dragging, you now have to *write* your code, but why?

What does this mean?

- Instead of clicking and dragging, you now have to *write* your code, but why?
  - It probably faster anyway :)

What does this mean?

- Instead of clicking and dragging, you now have to *write* your code, but why?
  - It probably faster anyway :)
  - The selection of blocks is so small you would run out quickly This way, we can define new ways to write or do something

What does this mean?

- Instead of clicking and dragging, you now have to *write* your code, but why?
  - It probably faster anyway :)
  - The selection of blocks is so small you would run out quickly This way, we can define new ways to write or do something
- On the other hand

What does this mean?

- Instead of clicking and dragging, you now have to *write* your code, but why?
  - It probably faster anyway :)
  - The selection of blocks is so small you would run out quickly This way, we can define new ways to write or do something
- On the other hand
  - Syntax implies syntax errors

What does this mean?

- Instead of clicking and dragging, you now have to *write* your code, but why?
  - It probably faster anyway :)
  - The selection of blocks is so small you would run out quickly This way, we can define new ways to write or do something
- On the other hand
  - Syntax implies syntax errors
  - Scope checking implies scoping errors

What does this mean?

- Instead of clicking and dragging, you now have to *write* your code, but why?
  - It probably faster anyway :)
  - The selection of blocks is so small you would run out quickly
    This way, we can define new ways to write or do something
- On the other hand
  - Syntax implies syntax errors
  - Scope checking implies scoping errors
  - Defining variables implies undefined variable errors. . .

Python IO

```
# blocks like Ask become input()
name = input("What is your name? ")

# the if block comes a multiline statement
if name == "Open Sesame":
    # blocks like Say become print()
    print("I don't think so!")
else:
    print("Hello " + name + "!")
```

Python flow

```python
# we can define new functions with def
def divide(x, y):
    # variable assignment simply becomes 'x = y'
    d = 0

    # 'while condition' loops keeps running until
    # 'condition' is false
    while x > y:
        # we can compose operators and assignments
        # this means that 'x -= y' is the same as
        # 'x = x - y'
        x -= y
        d += 1
```

Codewords

- Adapt your codeword program to Python
- Use a `while` loop to promote it to a login screen
    - Keep asking until correct

Higher or lower?

- Generate a random number between 1 and 100
- Let the user guess the number giving hints until correct
  - if the user guesses below the answer, print “higher”
  - if the user guesses above, print “lower”
  - if the user guesses correctly, print fireworks and applause