

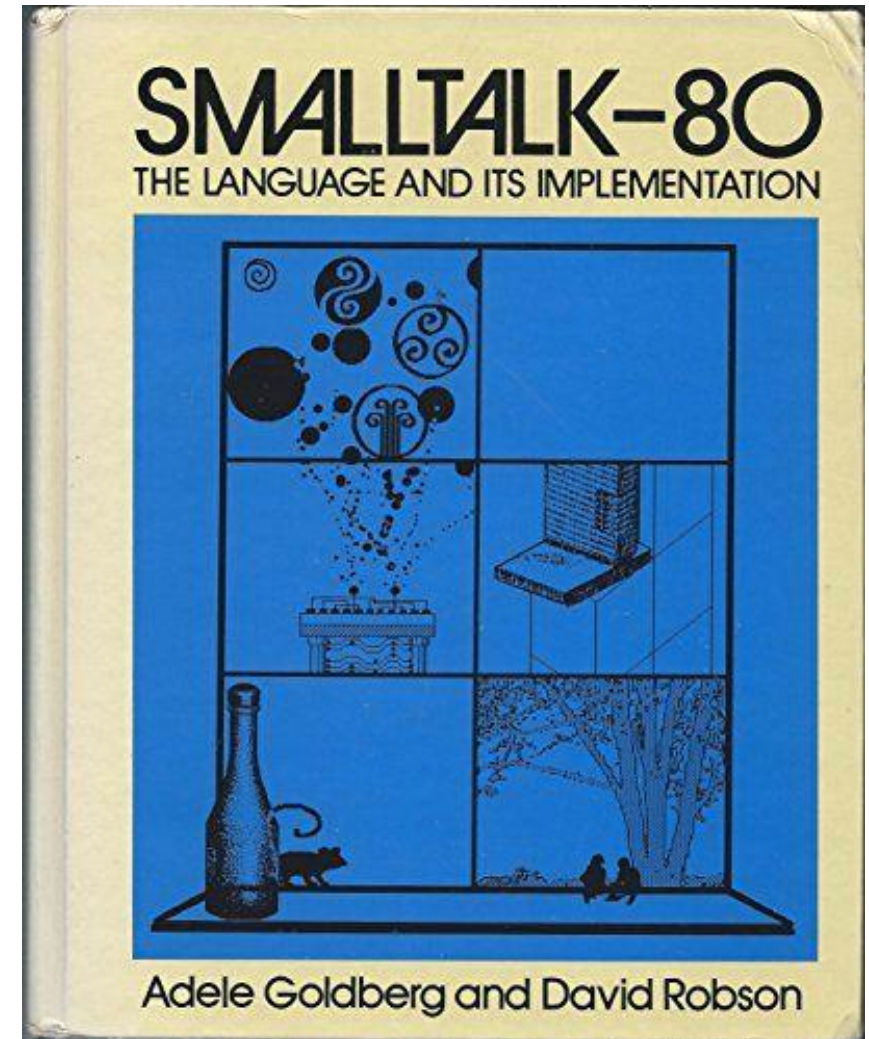


SMALLTALK

AHMED MOHAMED

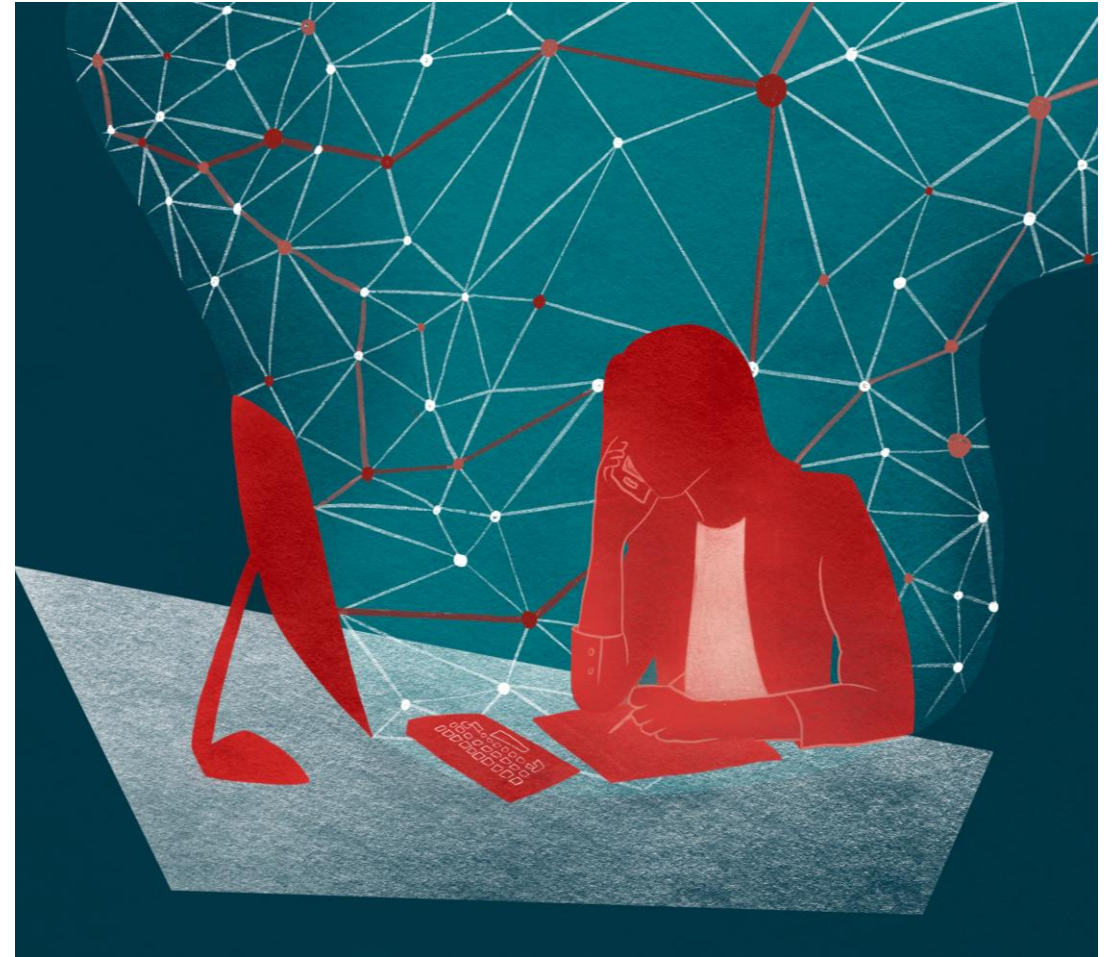
WHAT IS SMALLTALK

- Smalltalk is a fully object-oriented, dynamically typed, and reflective programming language
- It has no non-object types that is why it is called a fully OOP. This means everything in the language is an object.
- Strings are objects, numbers are object and even the classes are objects
- Created in the 1970s at Xerox PARC's Learning Research Group
- Smalltalk was designed as a live environment. It gives you the ability to change code instantly, inspect code in real time and explore ideas interactively.
- Built for education, constructionist learning, and interactive programming
- Main creators include Alan Kay, Dan Ingalls, Adele Goldberg, Ted Kaehler, Scott Wallace, and others



Why was Smalltalk a Big Deal

- Pushed the idea that everything is an object
- Introduced the first real IDE plus a browser-based development. This gave programmers a live coding environment and visual tools to see their class and methods.
- Messaging model heavily influenced modern OOP languages. This means that Smalltalk doesn't call functions everything is an object, so those objects send messages.
- Early example of metaprogramming and reflection. This means the program can view what structures it has and modify the methods while the system is running and create new classes dynamically.
- Smalltalk language and environment were influential in the history of the graphical user interface (GUI)
- Inspired later languages like Ruby, Python, Java, JavaScript



How Smalltalk code works

- Smalltalk has a very small, clean syntax.
- Almost no keywords needed everything is just an object and a message.
- Programs are built by objects sending messages to each other.
- Control flow (conditionals and loops) are also messages, not special syntax.
- Very readable, very predictable, easy to learn.
- Smalltalk uses . to end a statement.

```
1. 'Hello World' printNl.  
2. 3 factorial printNl.  
3.
```


Success #stdin #stdout 0.01s 7804KB

 comments (0)

 stdin

 copy

Standard input is empty

 stdout

 copy

'Hello World'

6

The Types of Smalltalk Messages

- They are three types of Smalltalk messages:

1 unary messages. It usually is a single symbol that maybe many words joined together in camel case like `BigArrayLength`.

2 Binary messages. This are symbolic operators, they take one argument. Used in arithmetic operations. (Small talk doesn't have arithmetic precedence).

3 Keyword messages. This are multiple arguments using semicolons which separates the arguments lexically.

- Once you understand these three, you can read the whole language
- Smalltalk avoids special rules or operator precedence this makes it one of the simplest languages ever made.

```
1. x:= 10.
2. x println.
3.
4. ('hello' asUppercase) println. "Unary message"
5.
6. (10 + 5 * 2 / 3) println. "Binary message"
7.
8. items := Array new: 3. "Keyword message"
9. items at: 2 put: 'cat'.
10. items println.
11.
```


Success #stdin #stdout 0.01s 8012KB

 comments (0)

 stdin

 copy

Standard input is empty

 stdout

 copy

```
10
'HELLO'
10
(nil 'cat' nil )
```


MORE SYNTAX RULES

Smalltalk evaluates messages in this order:

- Unary
- Binary
- Keyword

One neat thing about Smalltalk is you can keep chaining messages. this makes a lot of things simple and quick.

Variables start with lowercase while Objects start with uppercase and that is how you can distinguish between them.

Smalltalk simplicity invites you to try new things and work through the intuitiveness of OOP.

```
1. "messages in order"
2. X:= 3+4 factorial.
3. X printNl.
4.
5. 'hello' asUppercase reverse printNl.
6.
7. words := Array new: 5.
8. words printNl.
9.
10.
```

Success #stdin #stdout 0.01s 8000KB

 comments (0)

 stdin

 copy

Standard input is empty

 stdout

 copy

27

'OLLEH'

(nil nil nil nil nil)

Conditionals in Smalltalk

- Smalltalk does not use traditional if statements like other languages. Instead, Boolean objects (true and false) directly decide what happens by receiving messages.
- Logic works by sending keyword messages such as `ifTrue:`, `ifFalse:`, and `ifTrue:ifFalse:`. These messages take blocks (anonymous functions) that contain the code to run.
- Conditionals feel more like decision messages rather than control structures. This fits Smalltalk's philosophy that objects control all behavior.
- Smalltalk conditionals are highly flexible because blocks can contain multiple expressions, return values, or modify variables.
- This makes the code very readable because it reads like normal human language. if this happens, do this, and then print.

```
1. x := 10.
2.
3. (x > 5)
4.     ifTrue: [ 'x is greater than 5' printNl ]
5.     ifFalse: [ 'x is NOT greater than 5' printNl ].
6.
7. (x = 10)
8.     ifTrue: [ 'x is exactly 10' printNl. 'Value of x^2 is: 'print. y :=(x*x). y printNl ].
9.
10. (x < 3)
11.     ifFalse: [ 'x is not actually less than 3' printNl ].
12.
```

Success #stdin #stdout 0.01s 7952KB

 comments (0)

 stdin

 copy

Standard input is empty

 stdout

 copy

'x is greater than 5'

'x is exactly 10'

'Value of x^2 is: '100

'x is not actually less than 3'

Loops and Iteration in Smalltalk

- Loops are implemented by sending messages to a numbers of collections.
- TimesRepeat runs a block a fixed number of times, thus providing simple repetition.
- whileTrue: or whileFalse: loops execute as long as the set condition remains in the current state.
- Collections can be iterated directly using messages like do: or to:do:. This gives us a way to deal with their data using functions which is used for sequential access.
- Iteration in Smalltalk is uniform: numbers, booleans, and collections all respond to the same style of messages, making loops consistent across them and it helps with the readability.
- Output uses Transcript show: which is a global console object that prints text or numbers consistently in Smalltalk. Unlike print you need ; cr to add a newline.

```
1. 5 timesRepeat: [ 'Looping Smalltalk!' printNl ].
2.
3. "Classic while Loop"
4. counter := 1.
5. [counter <= 5] whileTrue: [
6.     Transcript show: 'Counter is '; show: counter printString; cr.
7.     counter := counter + 1
8. ].
9.
10. "Iterating over a collection with i"
11. numbers := OrderedCollection new.
12. numbers add: 2; add: 4; add: 6.
13.
14. 1 to: numbers size do: [:i |
15.     Transcript show: 'Position '; show: i printString; cr.
16.     Transcript show: 'Value: '; show: (numbers at: i) printString; cr
17. ].
```

Success #stdin #stdout 0.01s 8636KB

 stdin

Standard input is empty

 stdout

```
'Looping Smalltalk!'
'Looping Smalltalk!'
'Looping Smalltalk!'
'Looping Smalltalk!'
'Looping Smalltalk!'
Counter is 1
Counter is 2
Counter is 3
Counter is 4
Counter is 5
Position 1
Value: 2
Position 2
Value: 4
Position 3
Value: 6
```


Smalltalk example

Summing all the Even Numbers in a Collection

```
1.  "Create a collection of numbers from 1 to 10"
2.  numbers := OrderedCollection new.
3.  1 to: 10 do: [:i | numbers add: i ].
4.
5.  "Initialize the total var"
6.  total := 0.
7.
8.  "Iterating over the collection and summing the even numbers"
9.  numbers do: [:n |
10.      (n \\ 2 = 0) ifTrue: [      "\\ integer divide"
11.          total := total + n
12.      ]
13.  ].
14.
15.  "Print the sum"
16.  Transcript show: 'Sum of even numbers: '; show: total printString; cr.
17.
```

Success #stdin #stdout 0.01s 8008KB

 comments (0)

 stdin

 copy

Standard input is empty

 stdout

 copy

Sum of even numbers: 30

Why didn't Smalltalk catch on

- During the late 1980s–mid 1990s, Smalltalk was commercialized by ParcPlace Systems and Digitalk.
- ParcPlace: focused on Unix/Sun systems.
- Digitalk: focused on Intel PCs running Windows or OS/2.
- Smalltalk faced challenges like high memory requirements, limited performance, and weak database connectivity.
- ParcPlace & Digitalk merged in 1995 into a merger called ObjectShare, but they still struggled to compete with Java.
- By 1999, ObjectShare dissolved; Cincom acquired VisualWorks; other vendors like GemTalk and Instantiations continue Smalltalk support.
- Open-source derivatives include Squeak, GNU Smalltalk, Pharo, are often used in educational and research environments.

STRENGTHS and WEAKNESSES OF Smalltalk

STRENGTH


- Fully object-oriented
- Easily readable syntax which resembles natural language.
- Blocks and closures allow users to be flexible and have reusable code.
- Live IDE and visualization early environment for interactive coding, class browsers, and debugging.
- Metaprogramming and reflection users can inspect and modify objects at runtime.
- Encourages uniformity and consistency across loops, collections, and logic.

WEAKNESSES

- No traditional arithmetic which means it is upto the user to make sure his precedence is correct.
- Dynamic typing reduces runtime performance.
- Hard to find what you are looking for. it is a small community, fewer libraries, and less commercial support.
- Harder to find resources compared to other languages.
- Requires 1 indexing awareness for arrays and collections which is different from most established languages.



Legacy, Modern Use and Future

- **Smalltalk pioneered a lot of great programming frameworks**
 - **Influenced Ruby, Python, Java, JavaScript, and other OOP languages.**
 - **Modern IDEs and live coding tools borrow heavily from Smalltalk.**
 - **Still used in niche environments like financial systems, research, and education.**
 - **Open-source versions: Squeak, Pharo, GNU Smalltalk.**
 - **Modern frameworks like Seaside & AIDA/Web enable the simplifications of complex web applications using smalltalk.**
 - **concepts like blocks, message-passing, and reflection continue to inspire programming paradigms.**
- 



QUESTIONS ?

<https://learnxinyminutes.com/smalltalk/>
<https://en.wikipedia.org/wiki/Smalltalk>