In defence of language as an interface
A statement of the obvious

Luca Saiu
https://ageinghacker.net
positron@gnu.org
GNU Project

GNU Hackers’ Meeting 2022
İzmir, Turkey
October 1st, October 2nd 2022
Hello, my name is Luca Saiu.

My web site is https://ageinghacker.net
About me

Hello, my name is Luca Saiu.

My web site is https://ageinghacker.net
Hello, my name is Luca Saiu.

My web site is https://ageinghacker.net
Look at me

Can you notice anything?
Look at me

Can you notice anything?
Look at me

Can you notice anything?

I am old! Young people usually disagree with me. I still think I am right.

Luca Saiu

https://ageinghacker.net

In defence of language as an interface

GHM 2022
Look at me

Can you notice anything?

I am old! Young people usually disagree with me. I still think I am right.

Luca Saiu
https://ageinghacker.net

In defence of language as an interface

GHM 2022
Look at me

Can you notice anything?

I am old! Young people usually disagree with me.
I still think I am right.
Look at me

Can you notice anything?

I am old! Young people usually disagree with me.
I still think I am right.

Luca Saiu
https://ageinghacker.net
Computers are beautiful and complex.

My claim

the best way of harnessing the power of computers is through a linguistic interface. No other way will be as effective.

I shall argue my case by showing you an example problem in detail.
Computers are beautiful and complex.

**My claim**

the best way of harnessing the power of computers is through a **linguistic interface**. No other way will be as effective.

I shall argue my case by showing you an example problem in detail.
Test case: generating thumbnails for a photo collection

I have many JPEG images in a directory tree under ~/pictures/.

For every directory $D$ in the tree directly containing pictures I want to make a new subdirectory of it named $D$/thumbs/ containing a scaled-down version of every picture directly in $D$. (For example if ~/pictures/foo/bar/quux.jpg exists then we want a thumbnail for it in ~/pictures/foo/bar/thumbs/: we can name the thumbnail file ~/pictures/foo/bar/thumbs/quux-thumb.jpg)

Assume that:
- every JPEG file has a name ending with “.jpg”, and every object with such name is actually a JPEG file.
- no object named thumbs/ exists in the tree at the beginning.
Test case: generating thumbnails for a photo collection

I have many JPEG images in a directory tree under ~/pictures/.

For every directory $D$ in the tree directly containing pictures I want to make a new subdirectory of it named $D/thumbs/$ containing a scaled-down version of every picture directly in $D$.

(For example if ~/pictures/foo/bar/quux.jpg exists then we want a thumbnail for it in ~/pictures/foo/bar/thumbs/: we can name the thumbnail file ~/pictures/foo/bar/thumbs/quux-thumb.jpg)

Assume that:

- every JPEG file has a name ending with “.jpg”, and every object with such name is actually a JPEG file.
- no object named thumbs/ exists in the tree at the beginning.
Test case: generating thumbnails for a photo collection

I have many JPEG images in a directory tree under ~/pictures/.

For every directory $D$ in the tree directly containing pictures I want to make a new subdirectory of it named $D/thumbs/$ containing a scaled-down version of every picture directly in $D$. (For example if ~/pictures/foo/bar/quux.jpg exists then we want a thumbnail for it in ~/pictures/foo/bar/thumbs/: we can name the thumbnail file ~/pictures/foo/bar/thumbs/quux-thumb.jpg)

Assume that:

- every JPEG file has a name ending with “.jpg”, and every object with such name is actually a JPEG file.
- no object named thumbs/ exists in the tree at the beginning.
Introduction

Test case: generating thumbnails for a photo collection

I have many JPEG images in a directory tree under ~/pictures/.

For every directory $D$ in the tree directly containing pictures I want to make a new subdirectory of it named $D$/thumbs/ containing a scaled-down version of every picture directly in $D$.

(For example if ~/pictures/foo/bar/quux.jpg exists then we want a thumbnail for it in ~/pictures/foo/bar/thumbs/: we can name the thumbnail file ~/pictures/foo/bar/thumbs/quux-thumb.jpg)

Assume that:

- every JPEG file has a name ending with “.jpg”, and every object with such name is actually a JPEG file.
- no object named thumbs/ exists in the tree at the beginning.
A Unix-style solution

We can solve the test-case problem with Bash.

```
luca@moore ~]$  
```
A Unix-style solution

We can solve the test-case problem with Bash.

```
[luca@moore ~]$ cd pictures
[luca@moore ~/pictures]$`

In defence of language as an interface GHM 2022
A Unix-style solution

We can solve the test-case problem with Bash.

```
[luca@moore ~]$ cd pictures
[luca@moore ~/pictures]$ for file in $(find -name '*.jpg'); do mkdir $(dirname "$file")/thumbs &> /dev/null; convert "$file" -scale 100 $(dirname "$file")/thumbs/$(basename "$file" .jpg)-thumb.jpg; done
```
Looking at the Bash command in detail

The same shell command, with more whitespace.

```bash
for file in $(find -name '*.jpg'); do
    mkdir $(dirname "$file")/thumbs &> /dev/null;
    convert "$file" -scale 100 $(dirname "$file")/thumbs/$(basename "$file" .jpg)-thumb.jpg;
done
```

No real change from the one-line version. What is the most important program being called in this command?
Looking at the Bash command in detail

The same shell command, with more whitespace.

```bash
for file in $(find -name '*.jpg'); do
    mkdir $(dirname "$file")/thumbs &> /dev/null;
    convert \
    "$file" \
    -scale 100 \
    $(dirname "$file")/thumbs/$(basename "$file" .jpg)-thumb.jpg;
done
```

The “heart” of this command is the program `convert`. Is there any other primitive program used here?
Looking at the Bash command in detail

The same shell command, with more whitespace.

```bash
for file in $(find -name '*.jpg'); do
    mkdir $(dirname "\$file"):thumbs &> /dev/null;
    convert "$file" -scale 100 \ $(dirname "$file"):thumbs/$(basename "$file" .jpg)-thumb.jpg;
done
```

The “heart” of this command is the program `convert`. Is there any other primitive program used here?
Looking at the Bash command in detail

The same shell command, with more whitespace.

```bash
for file in $(find -name '*.jpg'); do
    mkdir $(dirname "$file")/thumbs &> /dev/null;
    convert \
        "$file" \
        -scale 100 \ 
        $(dirname "$file")/thumbs/$(basename "$file" .jpg)-thumb.jpg;
done

... Several *other “primitive” programs* are run, and do an important job.
```
Looking at the Bash command in detail

The same shell command, with more whitespace.

```bash
for file in $(find -name '*.jpg'); do
    mkdir $(dirname "$file")/thumbs &> /dev/null;
    convert \
        "$file" \
        -scale 100 \n        $(dirname "$file")/thumbs/$(basename "$file" .jpg)-thumb.jpg;
done

Variables are names bound to values; here we use only one, but variables are an important linguistic feature.
```
The same shell command, with more whitespace.

```bash
for file in $(find -name '/*.jpg'); do
    mkdir $(dirname "$file")/thumbs &> /dev/null;
    convert \
        "$file" \
        -scale 100 \
        $(dirname "$file")/thumbs/$(basename "$file" .jpg)-thumb.jpg;
done
```

There are ways of combining commands to make larger commands: looping, sequencing, inserting the output of another command.
Making the command nicer

The command can be made more readable with more variable definitions.

```bash
for file in $(find -name '*.jpg'); do
directory=$(dirname "$file")/thumbs;
mkdir "$directory" &> /dev/null;
thumbfile="$directory/"$(basename "$file" .jpg)-thumb.jpg;
convert "$file" -scale 100 "$thumbfile";
done
```

Look how readable the `convert` invocation is now!

Are we happy with the command now? Let us make it reusable.
Making the command nicer

The command can be made more readable with more variable definitions.

```bash
for file in $(find -name '/*.jpg'); do
directory=$(dirname "$file")/thumbs;
mkdir "$directory" &> /dev/null;
thumbfile="$directory/"$(basename "$file" .jpg)-thumb.jpg;
convert "$file" -scale 100 "$thumbfile"
done

Look how readable the convert invocation is now!
```

Are we happy with the command now? Let us make it reusable.
Making the command nicer

The command can be made more readable with more variable definitions.

```bash
for file in $(find -name '/*.jpg'); do
directory=$(dirname "$file")/thumbs;
mkdir "$directory" &> /dev/null;
thumbfile="$directory/"$(basename "$file" .jpg)-thumb.jpg;
convert "$file" -scale 100 "$thumbfile";
done
```

Look how readable the `convert` invocation is now! Are we happy with the command now? Let us make it reusable.
Making the command nicer

The command can be made more readable with more variable definitions.

```bash
for file in $(find -name '*.jpg'); do
    directory=$(dirname "$file")/thumbs;
    mkdir "$directory" &> /dev/null;
    thumbfile="$directory/"$(basename "$file" .jpg)-thumb.jpg;
    convert \
        "$file" \n        -scale 100 \n        "$thumbfile";
done
```

Look how readable the `convert` invocation is now!
Are we happy with the command now? Let us make it reusable.
Now look carefully... 

Take the command...

```bash
for file in $(find -name '*.jpg'); do
directory=$(dirname "$file")/thumbs;
mkdir "$directory" &> /dev/null;
thumbfile="$directory/"$(basename "$file" .jpg)-thumb.jpg;
convert "$file" -scale 100 "$thumbfile";
done
```

Luca Saiu
https://ageinghacker.net
In defence of language as an interface GHM 2022
...Indent it a little to the right...

```bash
for file in $(find -name '*.jpg'); do
directory=$(dirname "$file")/thumbs;
mkdir "$directory" &> /dev/null;
thumbfile="$directory/"$(basename "$file" .jpg)-thumb.jpg;
convert "$file" -scale 100 "$thumbfile";
done
```
...And wrap it into a function.

```bash
make-thumbs-in () {
    cd "$1";
    for file in $(find -name '*.jpg'); do
        directory=$(dirname "$file")/thumbs;
        mkdir "$directory" &> /dev/null;
        thumbfile="$directory/"$(basename "$file" .jpg)-thumb.jpg;
        convert \
            "$file" \
            -scale 100 \ 
            "$thumbfile";
    done
}
```
Thanks to abstraction we have now added one new command in our language. We can just write:

```
make-thumbs-in /var/www/gallery
```

as if `make-thumbs-in` were an ordinary “primitive”.

Luca Saiu

https://ageinghacker.net

In defence of language as an interface GHM 2022
Every powerful language has three mechanisms [...]:

- **primitive expressions** which represent the simplest entities the language is concerned with,
- **means of combination**, by which compound elements are built from simpler ones, and
- **means of abstraction**, by which compound elements can be named and manipulated as units.

(This text is called “The Wizard Book”, after its cover picture; highly recommended. Look at the bibliography at the end.)

I claim that this characterisation must be extended to any computer-human interface.
Every powerful language has three mechanisms [...]:

- **primitive expressions** which represent the simplest entities the language is concerned with,
- **means of combination**, by which compound elements are built from simpler ones, and
- **means of abstraction**, by which compound elements can be named and manipulated as units.

(This text is called “The Wizard Book”, after its cover picture; highly recommended. Look at the bibliography at the end.)

I claim that this characterisation must be extended to any computer-human interface.
Every powerful language has three mechanisms [...]:

- **primitive expressions** which represent the simplest entities the language is concerned with,
- **means of combination**, by which compound elements are built from simpler ones, and
- **means of abstraction**, by which compound elements can be named and manipulated as units.

(This text is called “The Wizard Book”, after its cover picture; highly recommended. Look at the bibliography at the end.)

I claim that this characterisation must be extended to any computer-human interface.
Let us analyse languages

- Is **Bash** a “powerful language” according to the previous definition?
- What about **C**?
  - C has relatively weak primitives: is this a problem?
- What about **C++**?
- What about the **CPP** preprocessor?
- What about **Lisp**?
  ```lisp
  (doTIMES (i 10)
    (PROGN
      (MESSAGE "i is now %s" i)
      (SIT-FOR 1)))
  ```
Let us analyse languages

- Is **Bash** a “powerful language” according to the previous definition?
- What about **C**?
  - C has relatively weak primitives: is this a problem?
- What about **C++**?
- What about the **CPP** preprocessor?
- What about **Lisp**?

```lisp
(dotimes (i 10)
  (progn
    (message "i is now %s" i)
    (sit-for 1)))
```

Luca Saiu
https://ageinghacker.net

In defence of language as an interface GHM 2022
Let us analyse languages

- Is **Bash** a “powerful language” according to the previous definition?
- What about **C**?
  - C has relatively weak primitives: is this a problem?
- What about **C++**?
  - What about the CPP preprocessor?
  - What about Lisp?
    - `(do ((i 10)) ((*))
         (progn
           (message "i is now %s" i)
           (sit-for 1)))`
Let us analyse languages

- Is **Bash** a “powerful language” according to the previous definition?
- What about **C**?
  - C has relatively weak primitives: is this a problem?
- What about **C++**?
- What about the **CPP** preprocessor?
- What about **Lisp**?

```lisp
(dotimes (i 10)
  (prog
    (message "i is now \(\text{i}\)"
    (sit-for 1)))
```
Let us analyse languages

- Is **Bash** a “powerful language” according to the previous definition?
- What about **C**?
  - C has relatively **weak primitives**: is this a problem?
- What about **C++**?
- What about the **CPP** preprocessor?
- What about **Lisp**?

```lisp
(dotimes (i 10)
  (progn
    (message "i is now %s" i)
    (sit-for 1)))
```
Let us analyse languages

- Is **Bash** a “powerful language” according to the previous definition?
- What about **C**?
  - C has relatively weak primitives: is this a problem?
- What about **C++**?
- What about the **CPP** preprocessor?
- What about **Lisp**?

```lisp
(dotimes (i 10)
  (progn
    (message "i is now %s" i)
    (sit-for 1)))
```
Let us analyse languages

- Is **Bash** a “powerful language” according to the previous definition?
- What about **C**?
  - C has relatively **weak primitives**: is this a problem?
- What about **C++**?
- What about the **CPP** preprocessor?
- What about **Lisp**?

```
(dotimes (i 10)
  (progn
    (message "i is now %s" i)
    (sit-for 1)))
```
Compensating for weak primitives

If primitives are weak:

- given good abstraction we can build more powerful primitive-like features;
- If abstraction is insufficient we are stuck.

Of the three elements primitives are the least important: with sufficient power in abstraction and combination more powerful primitive-like elements can be rebuilt starting from very simple primitives.

Example: * can be defined as a function if you have +.
Example: ** can be defined as a function if you have *.
Compensating for weak primitives

If primitives are weak:

- given good abstraction we can build more powerful primitive-like features;
- If abstraction is insufficient we are stuck.

Of the three elements primitives are the least important: with sufficient power in abstraction and combination more powerful primitive-like elements can be rebuilt starting from very simple primitives.

Example: + can be defined as a function if you have *.
Example: * can be defined as a function if you have +.
Compensating for **weak primitives**

If primitives are weak:
- given good abstraction we can **build more powerful primitive-like features**;
- If abstraction is insufficient **we are stuck**.

Of the three elements primitives are the least important: with sufficient power in abstraction and combination more powerful primitive-like elements can be rebuilt starting from very simple primitives.

Examples:
- \( a \) can be defined as a function if you have \( + \).
- \( \times \) can be defined as a function if you have \( \cdot \).
If primitives are weak:

- given good abstraction we can build more powerful primitive-like features;
- If abstraction is insufficient we are stuck.

Of the three elements primitives are the least important: with sufficient power in abstraction and combination more powerful primitive-like elements can be rebuilt starting from very simple primitives.

Example: * can be defined as a function if you have +.
Example: ** can be defined as a function if you have *.
Compensating for weak primitives

If primitives are weak:

- given good abstraction we can build more powerful primitive-like features;
- If abstraction is insufficient we are stuck.

Of the three elements primitives are the least important: with sufficient power in abstraction and combination more powerful primitive-like elements can be rebuilt starting from very simple primitives.

Example: * can be defined as a function if you have +.
Example: ** can be defined as a function if you have *.
Compensating for weak primitives

If primitives are weak:

- given good abstraction we can **build more powerful primitive-like features**;
- If abstraction is insufficient we are stuck.

Of the three elements primitives are the least important: with sufficient power in abstraction and combination more powerful primitive-like elements can be rebuilt starting from very simple primitives.

Example: * can be defined as a function if you have +.
Example: ** can be defined as a function if you have *.
Compensating for *weak combinations* ... is in my opinion *impossible* (but very little is required).
Compensating for **weak combinations**

... is in my opinion *impossible* (but very little is required).
Compensating for weak combinations

... is in my opinion impossible (but very little is required).
Compensating for weak abstractions

... is impossible.
What about this?

- **primitives**: good! (Many programs doing complex things)
- **combination**: (sequential composition by hand?)
- **abstraction**: (is there any kind of macro?)

Luca Saiu  
https://ageinghacker.net  
In defence of language as an interface  
GHM 2022
What about this?

- **primitives**: good! (Many programs doing complex things)
- **combination**: (sequential composition by hand?)
- **abstraction**: (is there any kind of macro?)

Luca Saiu
https://ageinghacker.net
In defence of language as an interface
GHM 2022
primitives: good! (Many programs doing complex things)

combination: (sequential composition by hand?)

abstraction: (is there any kind of macro?)

Luca Saiu

https://ageinghacker.net

In defence of language as an interface

GHM 2022
primitives: **good!** (Many programs doing complex things)
combination (sequential composition by hand?)
abstraction (is there any kind of macro?)

Luca Saiu
https://ageinghacker.net
In defence of language as an interface
GHM 2022
What about this?

- **primitives**: good! (Many programs doing complex things)
- **combination**: (sequential composition by hand?)
- **abstraction**: (is there any kind of macro?)
primitives: **good!** (Many programs doing complex things)
combination: **(sequential composition by hand?)**
abstraction: **(is there any kind of macro?)** (Xnee?)
primitives: good! (Many programs doing complex things)
combination: (sequential composition \textbf{by hand}?)
abstraction: (is there any kind of macro?) (Xnee? [Hello Henrik])
primitives: good! (Many programs doing complex things)
combination: (sequential composition by hand?)
abstraction: (is there any kind of macro?) (Xnee? [Hello Henrik])

Luca Saiu
https://ageinghacker.net
In defence of language as an interface
GHM 2022
What about this?

The Emacs Editor

Emacs is the extensible, customizable, self-documenting real-time display editor. This manual describes how to edit with Emacs and some of the ways to customize it; it corresponds to GNU Emacs version 26.0.50.

If you are reading this in Emacs, type `\texttt{\textasciitilde h}` to read a basic introduction to the Info documentation system.

- **primitives**: A lot predefined functionality (interactive or Lisp)
- **combination**: (Lisp combination)
- **abstraction**: (Lisp)

Luca Saiu  
https://ageinghacker.net  
In defence of language as an interface  
GHM 2022
What about this?

The Emacs Editor

Emacs is the extensible, customizable, self-documenting real-time display editor. This manual describes how to edit with Emacs and some of the ways to customize it; it corresponds to GNU Emacs version 26.0.50.

If you are reading this in Emacs, type 'h' to read a basic introduction to the Info documentation system.

U:~%* info* (emacs) Top Top L9 (Info Narrow)

- **primitives**: A lot predefined functionality (interactive or Lisp)
- **combination**: (Lisp combination)
- **abstraction**: (Lisp)
What about this?

The Emacs Editor

Emacs is the extensible, customizable, self-documenting real-time display editor. This manual describes how to edit with Emacs and some of the ways to customize it; it corresponds to GNU Emacs version 26.0.50.

If you are reading this in Emacs, type ‘h’ to read a basic introduction to the Info documentation system.

U:%*- info* (emacs) Top Top L9 (Info Narrow)

- primitives: A lot predefined functionality (interactive or Lisp)
- combination (Lisp combination)
- abstraction (Lisp)
What about this?

primitives: A lot predefined functionality (interactive or Lisp)
- combination (Lisp combination)
- abstraction (Lisp)

Luca Saiu
https://ageinghacker.net
In defence of language as an interface
GHM 2022
What about this?

primitives: A lot predefined functionality (interactive or Lisp)
combination: (Lisp combination) (sequential composition in keyboard macros)
abstraction (Lisp) (real-time operation even without Lisp)

Luca Saiu
https://ageinghacker.net
In defence of language as an interface
GHM 2022
primitives: A lot predefined functionality (interactive or Lisp)
combination: (Lisp combination) (sequential composition in keyboard macros)
abstraction: (Lisp) (keyboard macros: even without Lisp!)
What about this?

The Emacs Editor

Emacs is the extensible, customizable, self-documenting real-time display editor. This manual describes how to edit with Emacs and some of the ways to customize it; it corresponds to GNU Emacs version 26.0.50.

If you are reading this in Emacs, type ‘\h’ to read a basic introduction to the Info documentation system.

primitives: A lot predefined functionality (interactive or Lisp)
combination: (Lisp combination) (sequential composition in keyboard macros)
abstraction: (Lisp) (keyboard macros: even without Lisp!)
What about this?

Emacs is the extensible, customizable, self-documenting real-time display editor. This manual describes how to edit with Emacs and some of the ways to customize it; it corresponds to GNU Emacs version 26.0.50.

If you are reading this in Emacs, type ‘h’ to read a basic introduction to the Info documentation system.

 primitives: A lot predefined functionality (interactive or Lisp)
 combination: (Lisp combination) (sequential composition in keyboard macros)
 abstraction: (Lisp) (keyboard macros: even without Lisp!)
What about this?

p≡p JSON Server Adapter - version: 0.22.3 “(48) Nohra”

1. Send example calls via jQuery

Server’s Address: http://127.0.0.1:4223/ja/0.1/
Security token: igpoYkd0JEDoFzxsXZ2_3SrFh0sTvc25ec6RqA0
Function name: encrypt_message

(Very hostile to free software: can you easily even run a modified version of JavaScript code from a web site?) (Of course apps are much worse)
(Very hostile to free software: can you easily even run a modified version of JavaScript code from a web site?)
What about this?

p≡p JSON Server Adapter - version: 0.22.3 “(48) Nohra”

1. Send example calls via jQuery

(Very hostile to free software: can you easily even run a modified version of JavaScript code from a web site?) (Of course apps are much worse)
What about **sign languages**?

- Non-textual but still languages, with a grammar. No expressivity problem.

Luca Saiu

https://ageinghacker.net

In defence of language as an interface

GHM 2022
What about sign languages?

- Non-textual but still languages, with a grammar! No expressivity problem.
What about **sign languages**?

- Non-textual but still **languages**, with a grammar! No expressivity problem.
What I mean by **language**

By language-interface I mean that a language phrase is expressed via **a term**:

```lisp
(dotimes (i 10)
  (progn
    (message "hello: i is %s" i)
    (sit-for 1)))
```

- The term encoding can be arbitrary and non-textual (for example a sign language or any other structured grammar of gestures or sounds)...

- ...But it must remain precise and formal.

In order to have acceptable power a language interface must include *all three* elements (primitives, combination, abstraction) at a sufficient level of sophistication.
What I mean by language

By language-interface I mean that a language phrase is expressed via a term:

(dotimes (i 10)
  (progn
    (message "hello: i is %s" i)
    (sit-for 1)))

- The term encoding can be arbitrary and non-textual (for example a sign language or any other structured grammar of gestures or sounds)...

- ...But it must remain precise and formal.

In order to have acceptable power a language interface must include all three elements (primitives, combination, abstraction) at a sufficient level of sophistication.
What I mean by **language**

By language-interface I mean that a language phrase is expressed via a **term**:

```
(dotimes (i 10)
  (progn
    (message "hello: i is %s" i)
    (sit-for 1)))
```

- The term **encoding can be arbitrary** and non-textual (for example a sign language or any other structured grammar of **gestures** or **sounds**)...

...But it must remain precise and formal.

In order to have acceptable power a language interface must include all three elements (primitives, combination, abstraction) at a sufficient level of sophistication.
What I mean by language

By language-interface I mean that a language phrase is expressed via a term:

```
(dotimes (i 10)
  (progn
    (message "hello: i is %s" i)
    (sit-for 1)))
```

- The term encoding can be arbitrary and non-textual (for example a sign language or any other structured grammar of gestures or sounds)...
- ...But it must remain precise and formal.

In order to have acceptable power a language interface must include all three elements (primitives, combination, abstraction) at a sufficient level of sophistication.
What I mean by **language**

By language-interface I mean that a language phrase is expressed via a **term**:

```lisp
(dotimes (i 10)
  (progn
    (message "hello: i is %s" i)
    (sit-for 1))
```

- The term **encoding can be arbitrary** and non-textual (for example a sign language or any other structured grammar of **gestures** or **sounds**)...
- ...But it must remain precise and formal.

In order to have acceptable power a language interface must include **all three** elements (**primitives**, **combination**, **abstraction**) at a sufficient level of sophistication.
I have spoken about movement and sound as ways to encode language terms.

What about pictures?
I have spoken about movement and sound as ways to encode language terms.

What about pictures?
What about picture languages?

**Scratch** *(only some versions of it are free software!)*. Intended for teaching programming to children.

Figure: Statements have an indentation at entry and a knob at exit; expressions are hexagons; complex statements have statement-shaped holes for sub-statements.

---

Luca Saiu

https://ageinghacker.net

In defence of language as an interface

GHM 2022
What about picture languages?

**Scratch** *(only some versions of it are free software!)*. Intended for teaching programming to children.

*Figure: Statements* have an indentation at entry and a knob at exit; *expressions* are hexagons; *complex statements* have *statement-shaped holes* for sub-statements.

[Statements only exist in *structured* form *(one entry point, one exit point)*. The nesting metaphor does not extend to expressions, which is an arbitrary limitation.]
What about picture languages?

**Scratch** (only some versions of it are free software!). Intended for teaching programming to children.

![Diagram of Scratch blocks](image)

**Figure:** Statements have an indentation at entry and a knob at exit; expressions are hexagons; complex statements have statement-shaped holes for sub-statements.

[Statements only exist in structured form (one entry point, one exit point). The nesting metaphor does not extend to expressions, which is an arbitrary limitation.]
Scratch (only some versions of it are free software!). Intended for teaching programming to children.
What about picture languages?

**Scratch** *(only some versions of it are free software!)*. Intended for teaching programming to children.

```
(whenever (clicked flag)
  (set! score 5)
  (forever
    (if (touching? coin-sprite)  
      (incr! score))))
```

My opinion: useless, solves a non-problem. Teach Lisp instead.

*(Scratch has combinations. Not sure about abstraction.)*
What about picture languages?

**Scratch** *(only some versions of it are free software!)*. Intended for teaching programming to children.

```plaintext
(whenever (clicked flag)
  (set! score 5)
  (forever
    (if (touching? coin-sprite)
      (incr! score)))))
```

My opinion: useless, solves a non-problem. Teach Lisp instead.

(Scratch has combinations. Not sure about abstraction.)
What about picture languages?

**Scratch** *(only some versions of it are free software!)*. Intended for teaching programming to children.

```plaintext
(whenever (clicked flag)
  (set! score 5)
  (forever
   (if (touching? coin-sprite)
     (incr! score))))
```

My opinion: **useless**, solves a non-problem. Teach Lisp instead.

(Scratch has combinations. Not sure about abstraction.)
What about picture languages?

**Scratch** *(only some versions of it are free software!)*. Intended for teaching programming to children.

```plaintext
(whenever (clicked flag)
  (set! score 5)
  (forever
    (if (touching? coin-sprite)
        (incr! score))))
```

My opinion: useless, solves a non-problem. Teach Lisp instead.

*(Scratch has combinations. Not sure about abstraction.)*
Focus on **non-interactive** programs

If I have time: interactive versus non-interactive.

**Non-interactive language phrases** are much easier to compose and abstract.
If we have time: the hardware human interface can limit the possible software interfaces.
What about this?

You can already imagine my opinion about this interface...

...I have a separate set of slides about Replicant, with other considerations.

Thanks for now
What about this?

You can already imagine my opinion about this interface...

...I have a separate set of slides about Replicant, with other considerations.

Thanks for now
What about this?

You can already imagine my opinion about this interface... 

... I have a separate set of slides about Replicant, with other considerations.

Thanks for now
What about this?

You can already imagine my opinion about this interface... 

... I have a separate set of slides about Replicant, with other considerations.

Thanks for now
Image credits

- Mate screenshot: Copyright © Tomskyhaha, released under CC BY-SA 4.0 [https://en.wikipedia.org/wiki/Graphical_user_interface#/media/File:Ubuntu_Mate_18.04.1_with_MATE_1.20.1.png](https://en.wikipedia.org/wiki/Graphical_user_interface#/media/File:Ubuntu_Mate_18.04.1_with_MATE_1.20.1.png)


- p≡p JSON Server Adapter web interface screenshot: Copyright © p≡p Foundation, software released under the GPL


- Scratch screenshots: Copyright © Jess Weichler, released under CC BY-SA 4.0

Luca Saiu

https://ageinghacker.net

In defence of language as an interface

GHM 2022