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

About me

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`

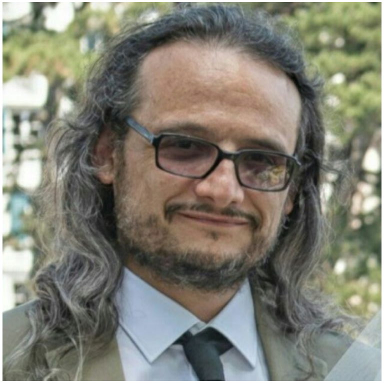
About me

Hello, my name is Luca Saiu.

My web site is `https://ageinghacker.net`

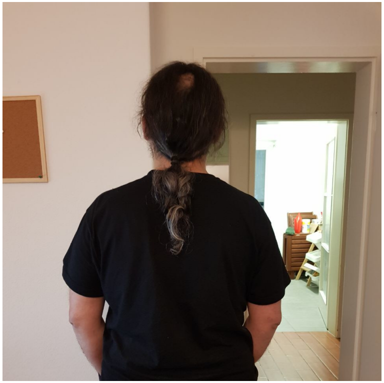
Look at me

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.

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**.

Look at me

Can you notice anything?

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

My claim for this presentation

Computers are beautiful and complex.

My claim

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

I shall argue my case by showing you an example problem in detail.

My claim for this presentation

Computers are beautiful and complex.

My claim

the best way of harnessing the power of computers is trough 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.

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]$
```

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.

```
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.

```
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.

```
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.

```
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.

```
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
```

Variables are names bound to values; here we use only one, but variables are an important linguistic feature.

Looking at the Bash command in detail

The same shell command, with more whitespace.

```
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.

```
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.

```
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.

```
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.

```
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.

Now look carefully. . .

Take the command. . .

```
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
```

Now look carefully. . .

. . . Indent it a little to the right. . .

```
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
```

This is called abstraction

. . . And wrap it into a function.

```
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
}
```

With abstraction we make new “primitive” commands

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”.

From Structure and Interpretation of Computer Programs

[Abelson et al., 1996] §1.1 {"The Elements of Programming"}

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.

From Structure and Interpretation of Computer Programs

[Abelson et al., 1996] §1.1 {“The Elements of Programming”}

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.

From *Structure and Interpretation of Computer Programs*

[Abelson et al., 1996] §1.1 {"The Elements of Programming"}

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?

```
(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)))
```

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)))
```

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)))
```

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)))
```

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)))
```

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.

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 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).

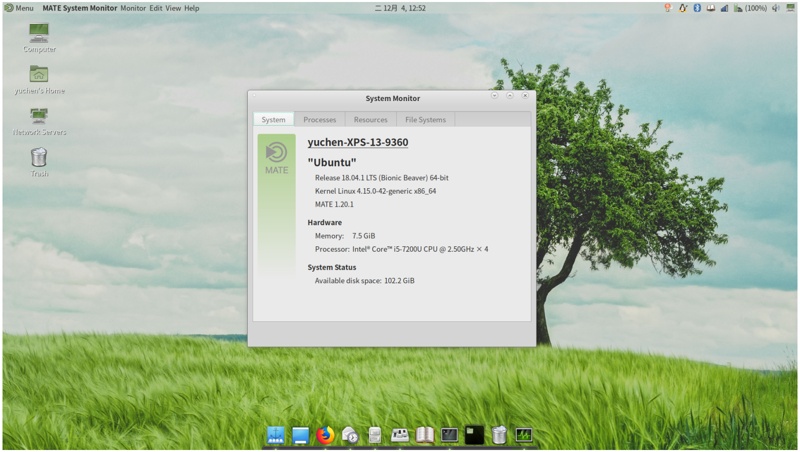
Compensating for weak combinations

. . . is in my opinion impossible (but very little is required).

Compensating for weak abstractions

. . . is impossible.

What about this?



yuchen-XPS-13-9360

"Ubuntu"

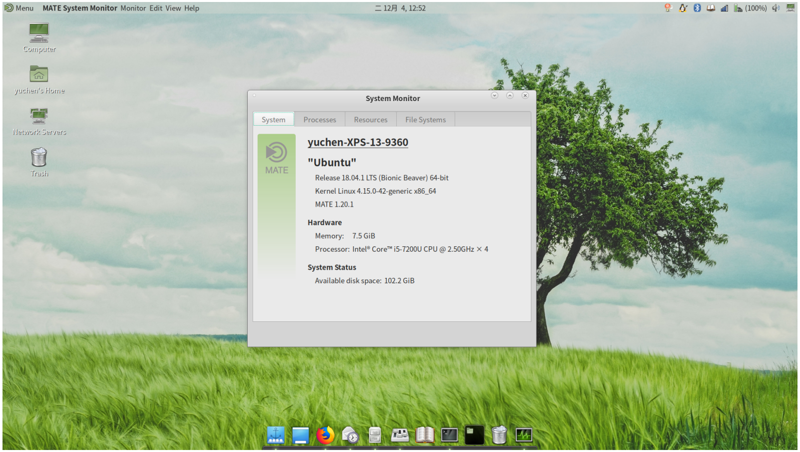
Release 18.04.1 LTS (Bionic Beaver) 64-bit
Kernel Linux 4.15.0-42-generic x86_64
MATE 1.20.1

Hardware
Memory: 7.5 GiB
Processor: Intel® Core™ i5-7200U CPU @ 2.50GHz × 4

System Status
Available disk space: 102.2 GiB

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

What about this?



yuchen-XPS-13-9360

"Ubuntu"

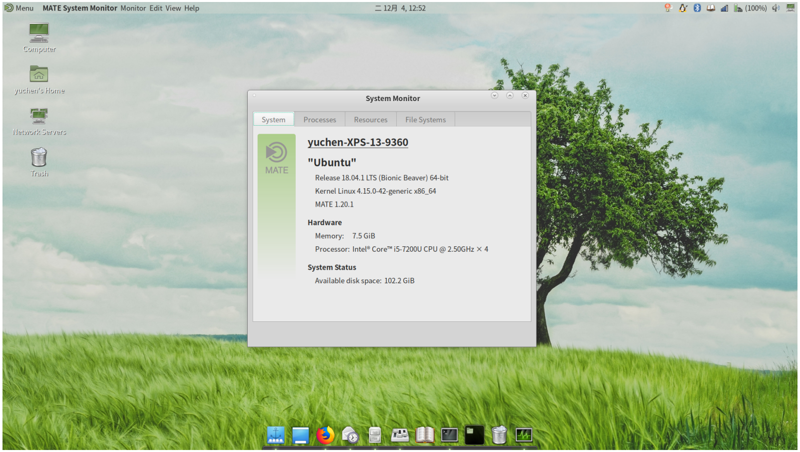
Release 18.04.1 LTS (Bionic Beaver) 64-bit
Kernel Linux 4.15.0-42-generic x86_64
MATE 1.20.1

Hardware
Memory: 7.5 GiB
Processor: Intel® Core™ i5-7200U CPU @ 2.50GHz × 4

System Status
Available disk space: 102.2 GiB

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

What about this?



yuchen-XPS-13-9360

"Ubuntu"

Release 18.04.1 LTS (Bionic Beaver) 64-bit

Kernel Linux 4.15.0-42-generic x86_64

MATE 1.20.1

Hardware

Memory: 7.5 GiB

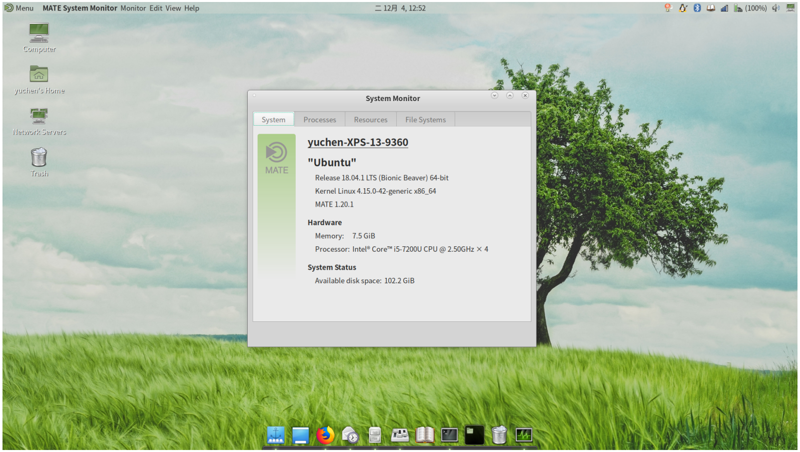
Processor: Intel® Core™ i5-7200U CPU @ 2.50GHz × 4

System Status

Available disk space: 102.2 GiB

- primitives: good! (Many programs doing complex things)
- combination: (sequential composition **by hand**?)
- abstraction: (is there any kind of macro?) (None?)

What about this?



yuchen-XPS-13-9360

"Ubuntu"

Release 18.04.1 LTS (Bionic Beaver) 64-bit
Kernel Linux 4.15.0-42-generic x86_64
MATE 1.20.1

Hardware

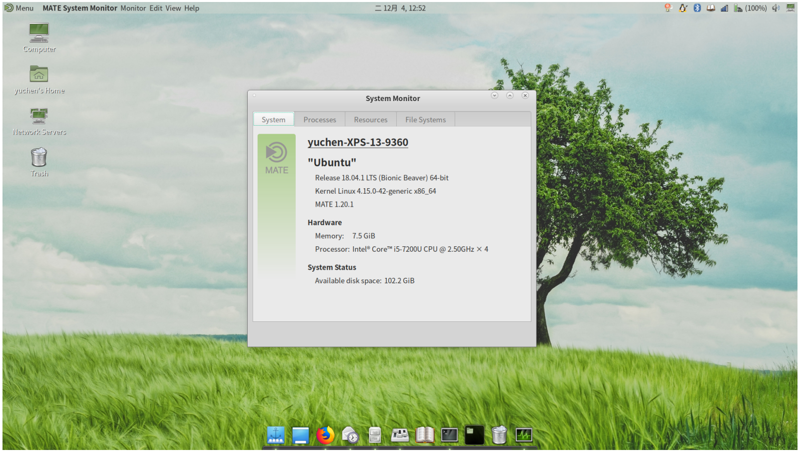
Memory: 7.5 GiB
Processor: Intel® Core™ i5-7200U CPU @ 2.50GHz × 4

System Status

Available disk space: 102.2 GiB

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

What about this?



yuchen-XPS-13-9360

"Ubuntu"

Release 18.04.1 LTS (Bionic Beaver) 64-bit
Kernel Linux 4.15.0-42-generic x86_64
MATE 1.20.1

Hardware

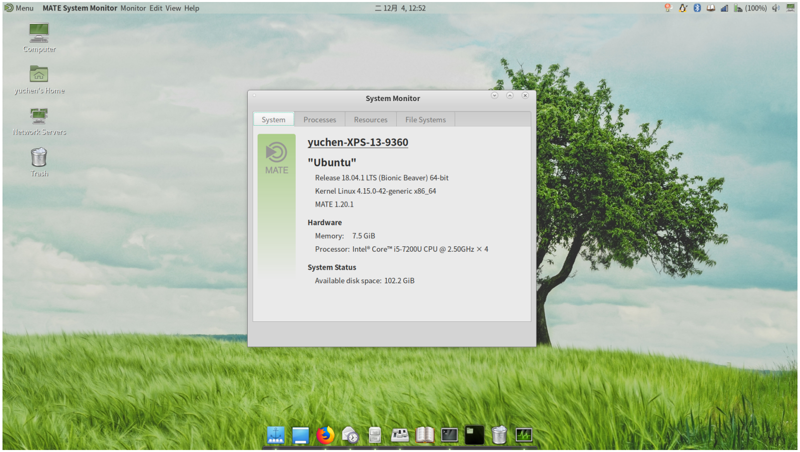
Memory: 7.5 GiB
Processor: Intel® Core™ i5-7200U CPU @ 2.50GHz × 4

System Status

Available disk space: 102.2 GiB

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

What about this?



yuchen-XPS-13-9360

"Ubuntu"

Release 18.04.1 LTS (Bionic Beaver) 64-bit

Kernel Linux 4.15.0-42-generic x86_64

MATE 1.20.1

Hardware

Memory: 7.5 GiB

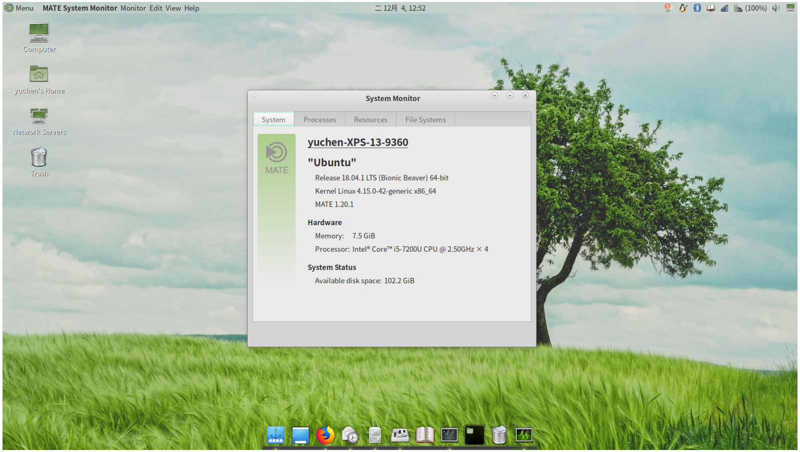
Processor: Intel® Core™ i5-7200U CPU @ 2.50GHz × 4

System Status

Available disk space: 102.2 GiB

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

What about this?



yuchen-XPS-13-9360

"Ubuntu"

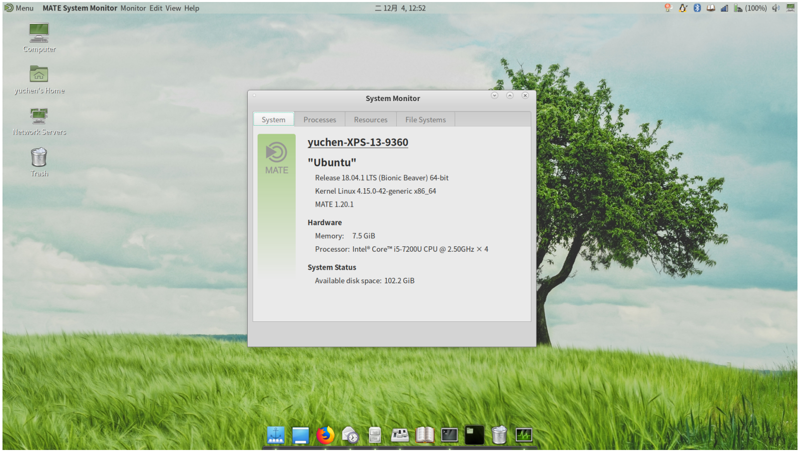
Release 18.04.1 LTS (Bionic Beaver) 64-bit
Kernel Linux 4.15.0-42-generic x86_64
MATE 1.20.1

Hardware
Memory: 7.5 GiB
Processor: Intel® Core™ i5-7200U CPU @ 2.50GHz × 4

System Status
Available disk space: 102.2 GiB

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

What about this?



yuchen-XPS-13-9360

"Ubuntu"

Release 18.04.1 LTS (Bionic Beaver) 64-bit
Kernel Linux 4.15.0-42-generic x86_64
MATE 1.20.1

Hardware

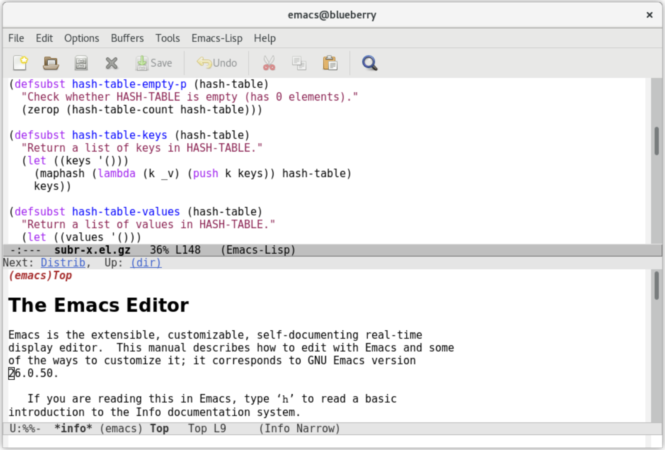
Memory: 7.5 GiB
Processor: Intel® Core™ i5-7200U CPU @ 2.50GHz × 4

System Status

Available disk space: 102.2 GiB

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

What about this?



```
(defsubst hash-table-empty-p (hash-table)
  "Check whether HASH-TABLE is empty (has 0 elements)."
  (zerop (hash-table-count hash-table)))

(defsubst hash-table-keys (hash-table)
  "Return a list of keys in HASH-TABLE."
  (let ((keys '()))
    (maphash (lambda (k _v) (push k keys)) hash-table)
    keys))

(defsubst hash-table-values (hash-table)
  "Return a list of values in HASH-TABLE."
  (let ((values '()))
-:---  subr-x.el.gz   36% L148   (Emacs-Lisp)
Next: Distrib,  Up: (dir)
(emacs)Top
```

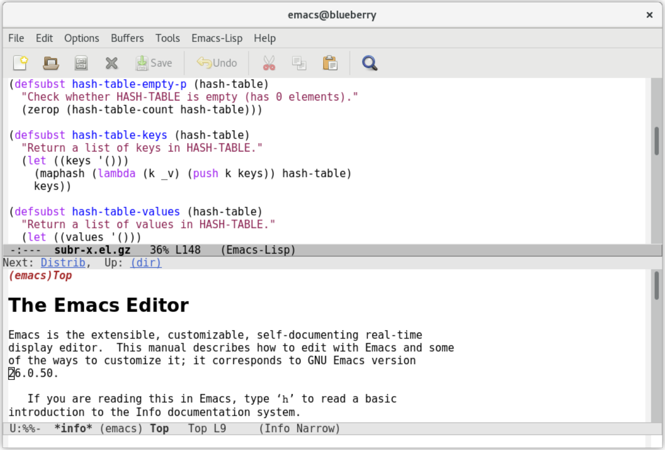
The Emacs Editor

```
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) (sequential composition in keyboard macros)
- abstraction: (Lisp) (keyboard macros: even without Lisp!)

What about this?



```
(defsubst hash-table-empty-p (hash-table)
  "Check whether HASH-TABLE is empty (has 0 elements)."
  (zerop (hash-table-count hash-table)))

(defsubst hash-table-keys (hash-table)
  "Return a list of keys in HASH-TABLE."
  (let ((keys '()))
    (maphash (lambda (k _v) (push k keys)) hash-table)
    keys))

(defsubst hash-table-values (hash-table)
  "Return a list of values in HASH-TABLE."
  (let ((values '()))
 -:---  subr-x.el.gz   36% L148    (Emacs-Lisp)
Next: Distrib,  Up: (dir)
(emacs)Top
```

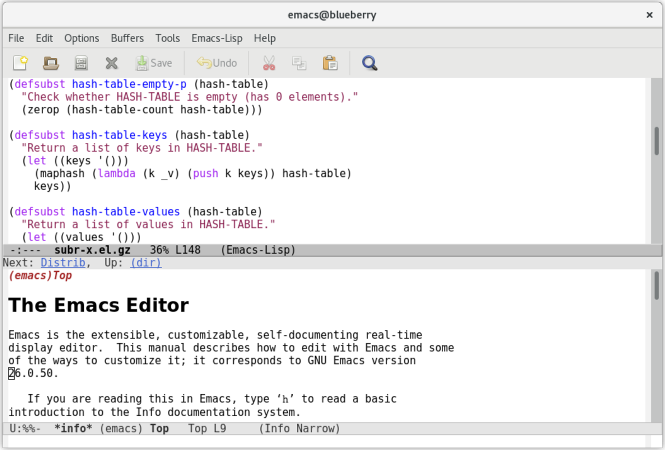
The Emacs Editor

```
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) (sequential composition in keyboard macros)
- abstraction: (Lisp) (keyboard macros, even without Lisp!)

What about this?



```
(defsubst hash-table-empty-p (hash-table)
  "Check whether HASH-TABLE is empty (has 0 elements)."
  (zerop (hash-table-count hash-table)))

(defsubst hash-table-keys (hash-table)
  "Return a list of keys in HASH-TABLE."
  (let ((keys '()))
    (maphash (lambda (k _v) (push k keys)) hash-table)
    keys))

(defsubst hash-table-values (hash-table)
  "Return a list of values in HASH-TABLE."
  (let ((values '()))
 -:---  subr-x.el.gz    36% L148    (Emacs-Lisp)
Next: Distrib,  Up: (dir)
(emacs)Top
```

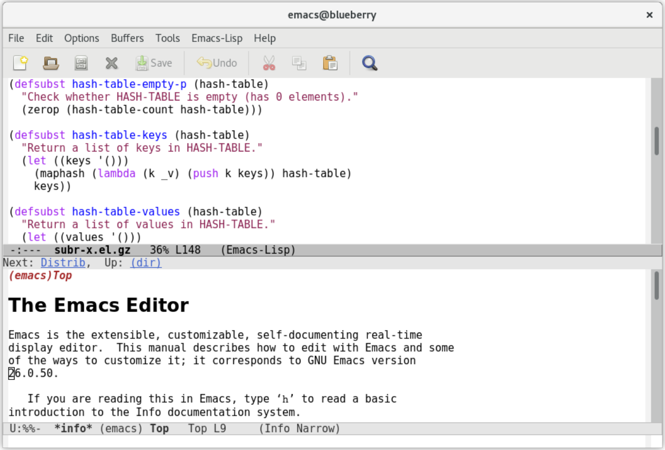
The Emacs Editor

```
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) (sequential composition in keyboard macros)
- abstraction: (Lisp) (keyboard macros: even without Lisp!)

What about this?



```
(defsubst hash-table-empty-p (hash-table)
  "Check whether HASH-TABLE is empty (has 0 elements)."
  (zerop (hash-table-count hash-table)))

(defsubst hash-table-keys (hash-table)
  "Return a list of keys in HASH-TABLE."
  (let ((keys '()))
    (maphash (lambda (k _v) (push k keys)) hash-table)
    keys))

(defsubst hash-table-values (hash-table)
  "Return a list of values in HASH-TABLE."
  (let ((values '()))
-:---  subr-x.el.gz   36% L148   (Emacs-Lisp)
Next: Distrib,  Up: (dir)
(emacs)Top
```

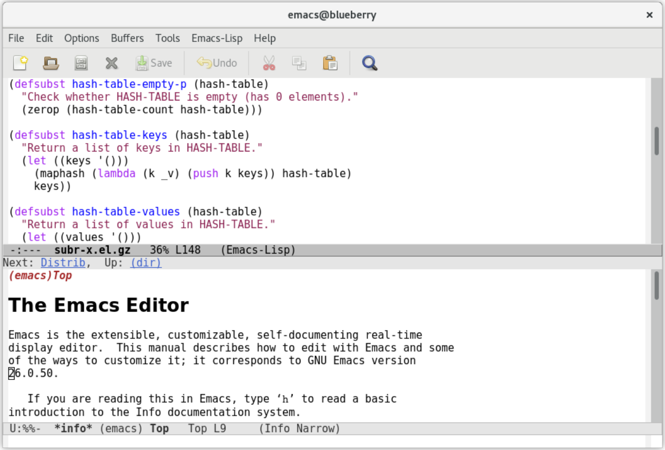
The Emacs Editor

```
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) (sequential composition in keyboard macros)
- abstraction: (Lisp) (keyboard macros: even without Lisp!)

What about this?



```
(defsubst hash-table-empty-p (hash-table)
  "Check whether HASH-TABLE is empty (has 0 elements)."
  (zerop (hash-table-count hash-table)))

(defsubst hash-table-keys (hash-table)
  "Return a list of keys in HASH-TABLE."
  (let ((keys '()))
    (maphash (lambda (k _v) (push k keys)) hash-table)
    keys))

(defsubst hash-table-values (hash-table)
  "Return a list of values in HASH-TABLE."
  (let ((values '()))
 -:---  subr-x.el.gz    36% L148    (Emacs-Lisp)
Next: Distrib,  Up: (dir)
(emacs)Top
```

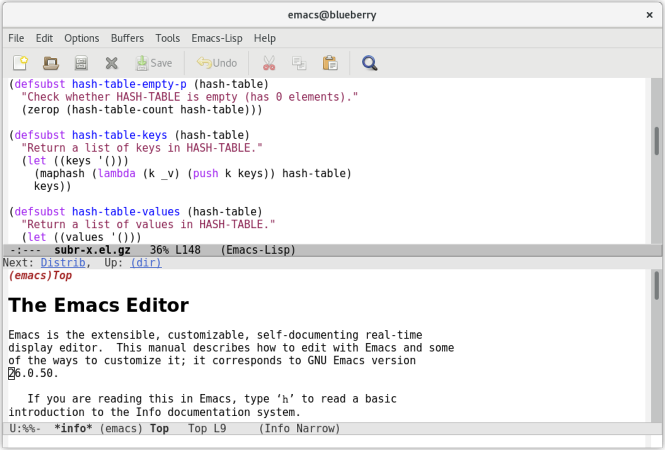
The Emacs Editor

```
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) (sequential composition in keyboard macros)
- abstraction: (Lisp) (keyboard macros: even without Lisp!)

What about this?



```
(defsubst hash-table-empty-p (hash-table)
  "Check whether HASH-TABLE is empty (has 0 elements)."
  (zerop (hash-table-count hash-table)))

(defsubst hash-table-keys (hash-table)
  "Return a list of keys in HASH-TABLE."
  (let ((keys '()))
    (maphash (lambda (k _v) (push k keys)) hash-table)
    keys))

(defsubst hash-table-values (hash-table)
  "Return a list of values in HASH-TABLE."
  (let ((values '()))
 -:---  subr-x.el.gz    36% L148   (Emacs-Lisp)
Next: Distrib,  Up: (dir)
(emacs)Top
```

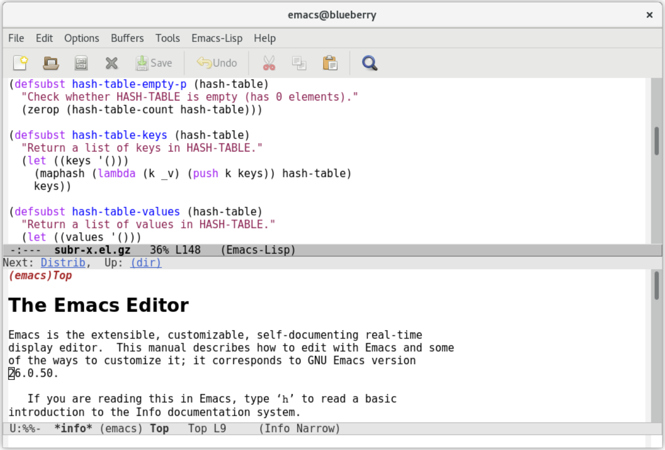
The Emacs Editor

```
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) (sequential composition in keyboard macros)
- abstraction: (Lisp) (keyboard macros: even without Lisp!)

What about this?



```
(defsubst hash-table-empty-p (hash-table)
  "Check whether HASH-TABLE is empty (has 0 elements)."
  (zerop (hash-table-count hash-table)))

(defsubst hash-table-keys (hash-table)
  "Return a list of keys in HASH-TABLE."
  (let ((keys '()))
    (maphash (lambda (k _v) (push k keys)) hash-table)
    keys))

(defsubst hash-table-values (hash-table)
  "Return a list of values in HASH-TABLE."
  (let ((values '()))
-:---  subr-x.el.gz    36% L148   (Emacs-Lisp)
Next: Distrib,  Up: (dir)
(emacs)Top
```

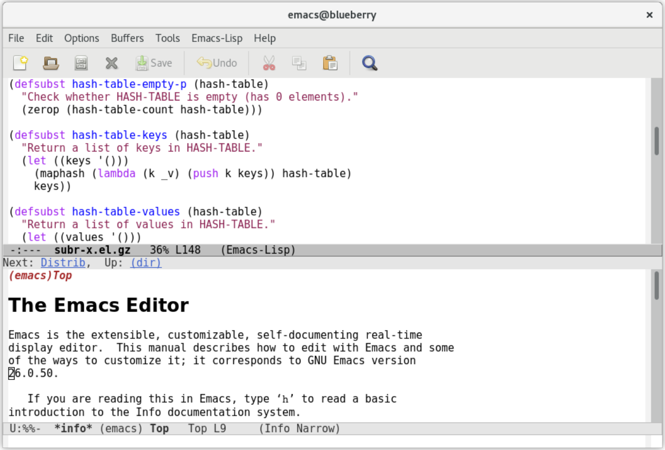
The Emacs Editor

```
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) (sequential composition in keyboard macros)
- abstraction: (Lisp) (keyboard macros: even without Lisp!)

What about this?



```
(defsubst hash-table-empty-p (hash-table)
  "Check whether HASH-TABLE is empty (has 0 elements)."
  (zerop (hash-table-count hash-table)))

(defsubst hash-table-keys (hash-table)
  "Return a list of keys in HASH-TABLE."
  (let ((keys '()))
    (maphash (lambda (k _v) (push k keys)) hash-table)
    keys))

(defsubst hash-table-values (hash-table)
  "Return a list of values in HASH-TABLE."
  (let ((values '()))
-:---  subr-x.el.gz   36% L148   (Emacs-Lisp)
Next: Distrib,  Up: (dir)
(emacs)Top
```

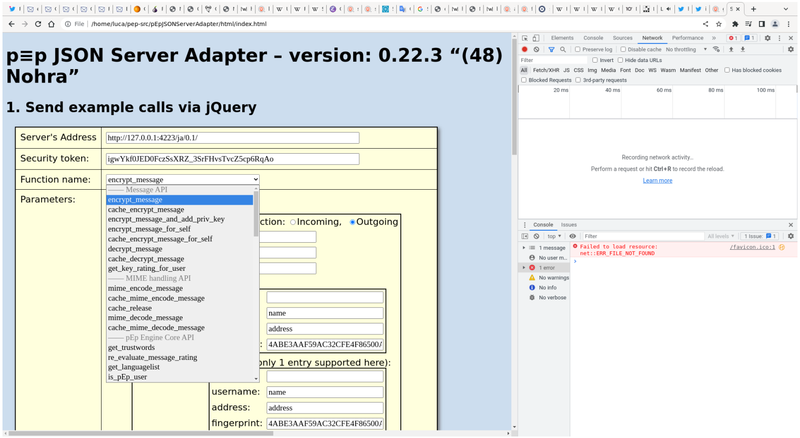
The Emacs Editor

```
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) (sequential composition in keyboard macros)
- abstraction: (Lisp) (keyboard macros: even without Lisp!)

What about this?


/home/luca/pep-src/pEpJSONServerAdapter/html/index.html

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:	igwYkf0JED0FczSsXRZ_3SrFHvsTvcZ5cp6RqAo
Function name:	encrypt_message
Parameters:	

—— Message API
encrypt_message
cache_encrypt_message
encrypt_message_and_add_priv_key
encrypt_message_for_self
cache_encrypt_message_for_self
decrypt_message
cache_decrypt_message
get_key_rating_for_user
—— MIME handling API
mime_encode_message
cache_mime_encode_message
cache_release
mime_decode_message
cache_mime_decode_message
—— pEp Engine Core API
get_trustwords
re_evaluate_message_rating
get_languagelist
is_pEp_user

ction: ○Incoming, [✓ Outgoing]

name
address
: 4ABE3AAF59AC32CFE4F86500/

only 1 entry supported here):

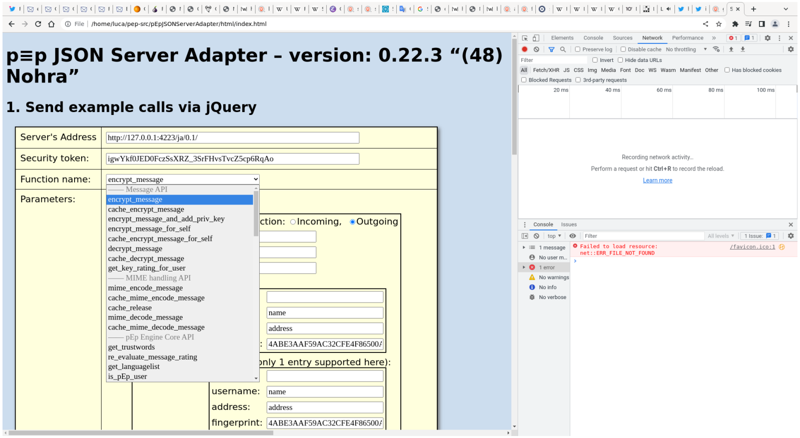
username: name
address: address
fingerprint: 4ABE3AAF59AC32CFE4F86500/

```
Recording network activity…
Perform a request or hit Ctrl+R to record the reload.
Learn more
```

```
1 message        Failed to load resource:                /favicon.ico:1
No user m...     net::ERR_FILE_NOT_FOUND
1 error
No warnings
No info
No verbose
```

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

What about this?



File | /home/luca/pep-src/pEpJSONServerAdapter/html/index.html

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:	igwYkf0JED0FczSsXRZ_3SrFHvsTvcZ5cp6RqAo
Function name:	encrypt_message
Parameters:	

—— Message API
encrypt_message
cache_encrypt_message
encrypt_message_and_add_priv_key
encrypt_message_for_self
cache_encrypt_message_for_self
decrypt_message
cache_decrypt_message
get_key_rating_for_user
—— MIME handling API
mime_encode_message
cache_mime_encode_message
cache_release
mime_decode_message
cache_mime_decode_message
—— pEp Engine Core API
get_trustwords
re_evaluate_message_rating
get_languagelist
is_pEp_user

ction: ○Incoming, [✓ Outgoing]

name
address
: 4ABE3AAF59AC32CFE4F86500/

nly 1 entry supported here):

username:	name
address:	address
fingerprint:	4ABE3AAF59AC32CFE4F86500/

Recording network activity...

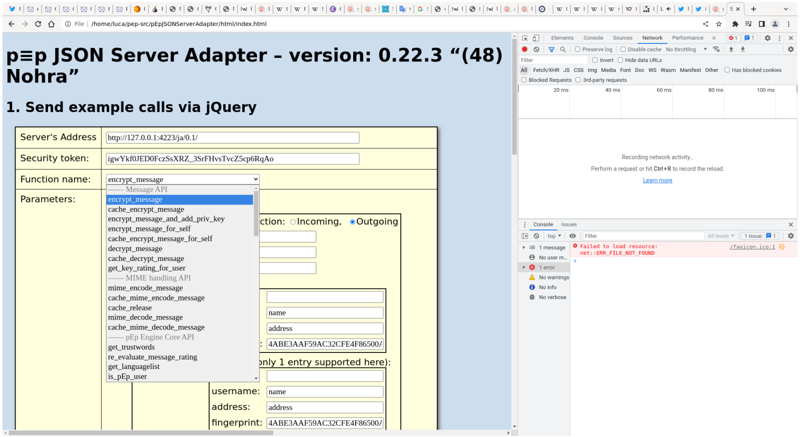
Perform a request or hit **Ctrl+R** to record the reload.

Learn more

```
Failed to load resource:                /favicon.ico:1
net::ERR_FILE_NOT_FOUND
```

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

What about this?



/home/luca/pep-src/pEpJSONServerAdapter/html/index.html

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:	igwYkf0JED0FczSsXRZ_3SrFHvsTvcZ5cp6RqAo
Function name:	encrypt_message
Parameters:	

—— Message API
encrypt_message
cache_encrypt_message
encrypt_message_and_add_priv_key
encrypt_message_for_self
cache_encrypt_message_for_self
decrypt_message
cache_decrypt_message
get_key_rating_for_user
—— MIME handling API
mime_encode_message
cache_mime_encode_message
cache_release
mime_decode_message
cache_mime_decode_message
—— pEp Engine Core API
get_trustwords
re_evaluate_message_rating
get_languagelist
is_pEp_user

ction: ○Incoming, ● Outgoing

name
address
: 4ABE3AAF59AC32CFE4F86500

nly 1 entry supported here):

username: name
address: address
fingerprint: 4ABE3AAF59AC32CFE4F86500

Elements Console Sources Network Performance
Preserve log Disable cache No throttling
Filter Invert Hide data URLs
All Fetch/XHR JS CSS Img Media Font Doc WS Wasm Manifest Other Has blocked cookies
Blocked Requests 3rd-party requests
20 ms 40 ms 60 ms 80 ms 100 ms

Recording network activity…
Perform a request or hit **Ctrl+R** to record the reload.
Learn more

Console Issues
top Filter All levels 1 Issue: 1
1 message
No user m…
1 error
No warnings
No info
No verbose

```
Failed to load resource:                    /favicon.ico:1
net::ERR_FILE_NOT_FOUND
```

(Very hostile to free software: can you easily even *run* a modified version of JavaScipt 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

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:

```
(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 [illegible])...
- ...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:

```
(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.

Other non-textual languages

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

What about pictures?

Other non-textual languages

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.

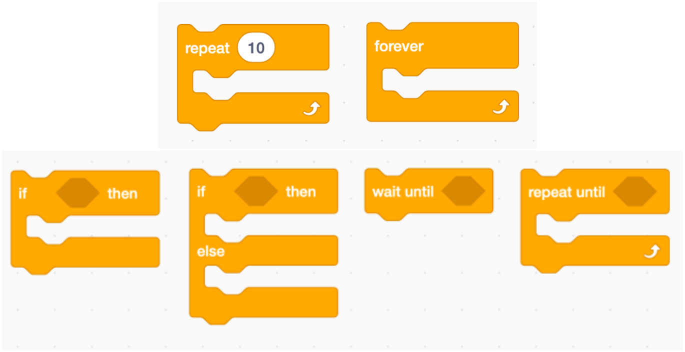


Figure: Statements have an indentation at entry and a knob at exit; expressions are hexagons; complex statements have statement-shaped holes for sub-statements.

[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.

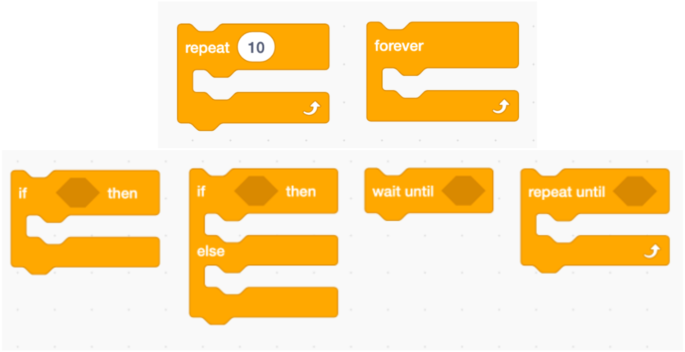


Figure: Statements have an indentation at entry and a knob at exit; expressions are hexagons; complex statements have statement-shaped holes for sub-statements.

[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.

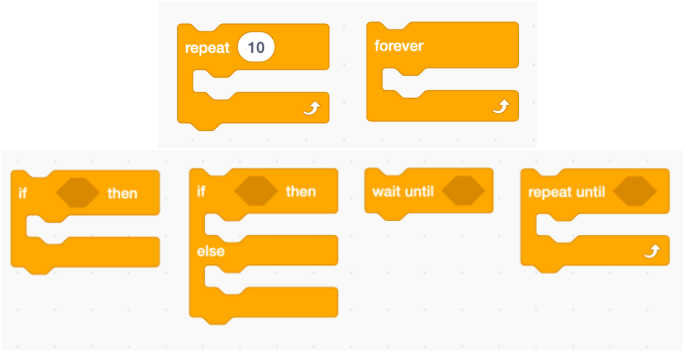
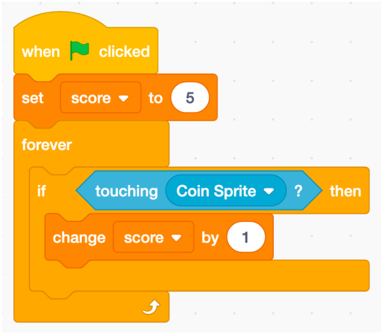


Figure: Statements have an indentation at entry and a knob at exit; expressions are hexagons; complex statements have statement-shaped holes for sub-statements.

[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.]

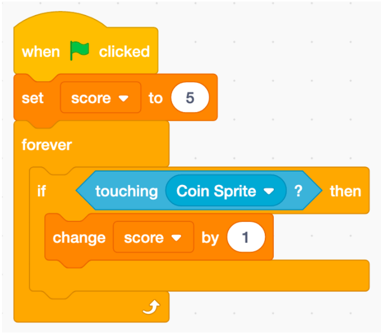
What about picture languages?

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


```
when [flag] clicked
set score to 5
forever
    if touching Coin Sprite ? then
        change score by 1
```

What about picture languages?

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



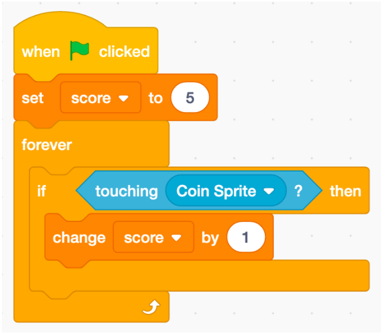
```
(whenever (clicked flag)
  (set! score 5)
  (forever
    (if (touching? coin-sprite)
      (incr! score))))
```

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.



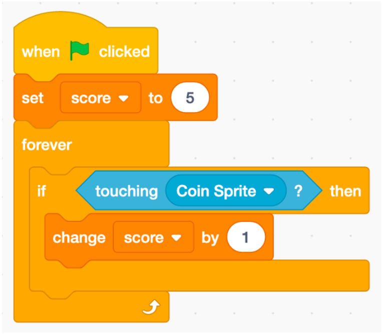
```
(whenever (clicked flag)
  (set! score 5)
  (forever
    (if (touching? coin-sprite)
      (incr! score))))
```

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.



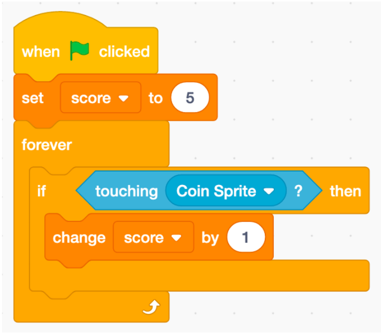
```
(whenever (clicked flag)
  (set! score 5)
  (forever
    (if (touching? coin-sprite)
      (incr! score))))
```

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.



```
(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

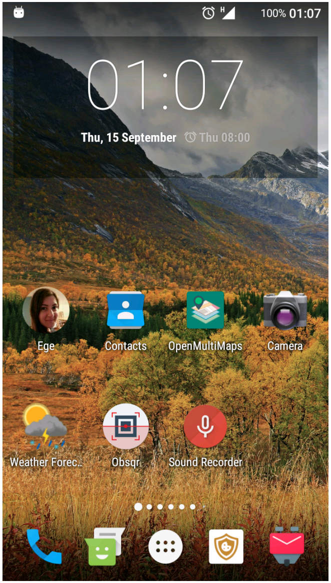
If I have time: interactive versus non-interactive.

Non-interactive language phrases are much easier to compose and abstract.

If we have time: hardware human interface

If we have time: the hardware human interface can limit the possible software interfaces.

What about this?

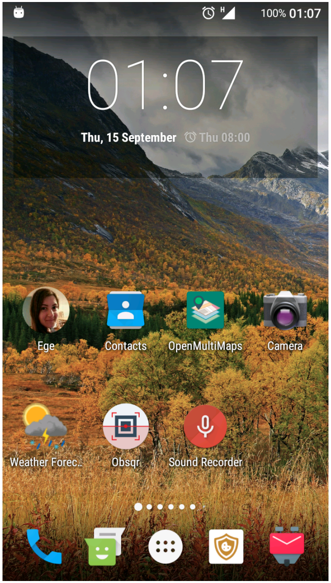


You can already imagine my opinion about this interface. . .

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

Thanks for now

What about this?

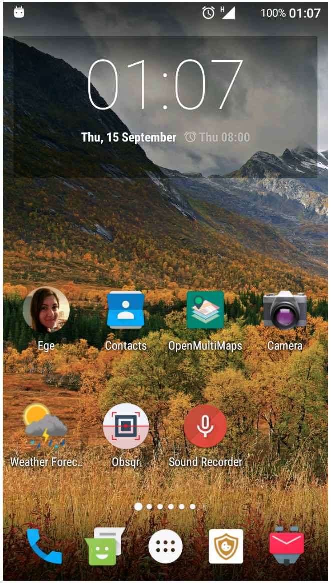


You can already imagine my opinion about this interface...

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

Thanks for now

What about this?

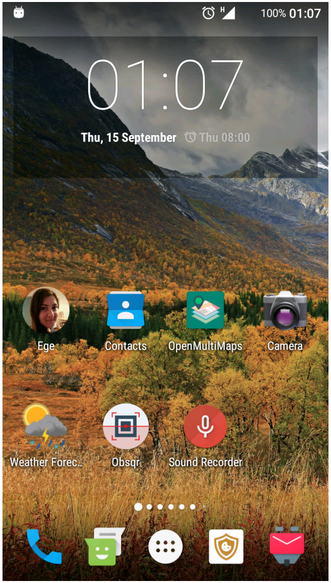


You can already imagine my opinion about this interface. . .

. . . 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 I

- 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
- Sign language: Copyright © 2008 David Fulmer, released under CC BY 2.0 https://en.wikipedia.org/wiki/American_Sign_Language#/media/File:ASL_family.jpg
- p≡p JSON Server Adapter web interface screenshot: Copyright © p≡p Foundation, software released under the GPL
- Emacs screenshot: Copyright © 2015-2022 Free Software Foundation, released under CC-BY-SA, from https://www.gnu.org/software/emacs
- Scratch screenshots: Copyright © Jess Weichler, released under CC BY-SA 4.0

Bibliography I

Abelson, H., Sussman, G. J., and Sussman, J. (1996). *Structure and Interpretation of Computer Programs*. MIT Press, second edition. The book is freely downloadable at https://cloudflare-ipfs.com/ipfs/QmQ3C4ooSCmBMuK7mKq4sqVAfGq9y5EJpWNGVTQzC1FRms?filename=sicp.pdf. See also the video lectures by the authors at https://ocw.mit.edu/courses/6-001-structure-and-interpretation-of-computer-programs-spri video_galleries/video-lectures/ following the first edition of the book.