CS51: Abstraction and Design in Computation

2: Fundamentals of functional languages

Now playing:
"Bucephalus Bouncing Ball"
Aphex Twin
Come To Daddy

John A. Paulson School of Engineering and Applied Sciences
Harvard University

expression *type*

meaning *function*

structure *definition*

value *abstraction*

expression:

expression: a meaningful combination of symbols

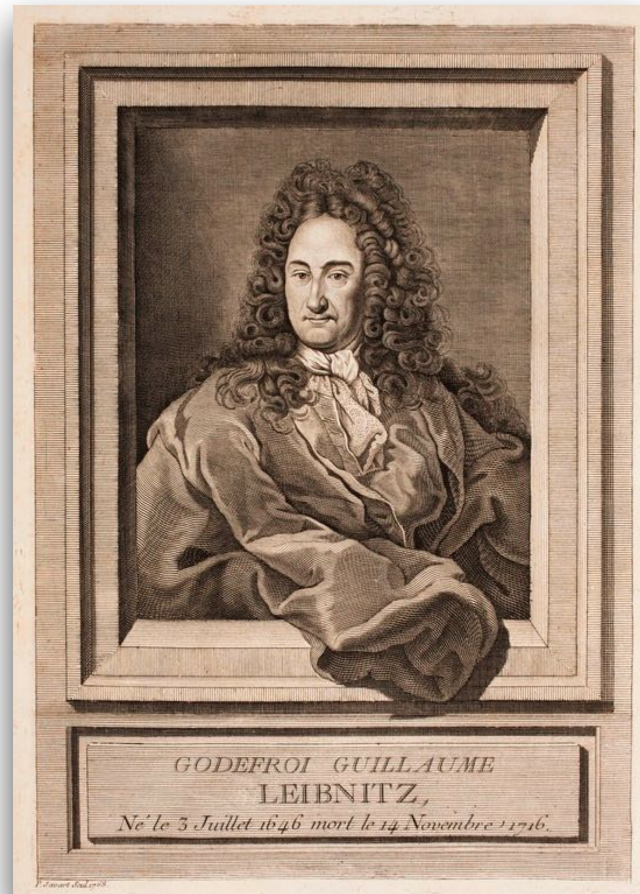
meaning:

meaning: that aspect of expressions the identity of which preserves truth under substitution

The indiscernibility of identicals

That *A* is the same as *B* signifies that the one can be substituted for the other, *salva veritate,* in any proposition whatever.

– Gottfried Wilhelm Leibniz



Gottfried Wilhelm Leibniz

```
3 + 4 + 5
```

$$\underline{3 + 4} + 5$$
$$\rightarrow 7 + 5$$

```
3 + 4 + 5
  ➡ 7 + 5
  ➡ 12
```

structure:

structure: the arrangement of parts in a complex whole

```
let rec gcd_euclid a b =
  if b = 0
  then a
  else gcd_euclid b (a mod b) ;;
```

```
let rec gcd_euclid a b = if b = 0
then a else gcd_euclid b (a mod b) ;;
```

```
3 + 4 * 5
```

```
3 + 4 * 5
➡ 7 * 5
```

```
3 + 4 * 5
  ➡ 7 * 5
  ➡ 35
```

3 + 4 * 5

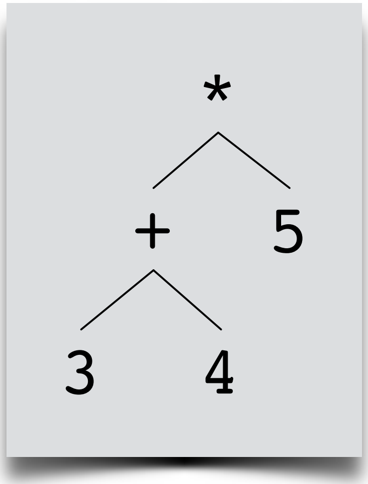
➡ 7 * 5

➡ 35

~~3 + 4 * 5~~

~~→ 7 * 5~~

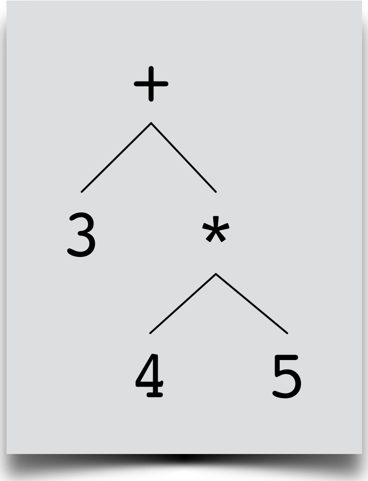
~~→ 35~~

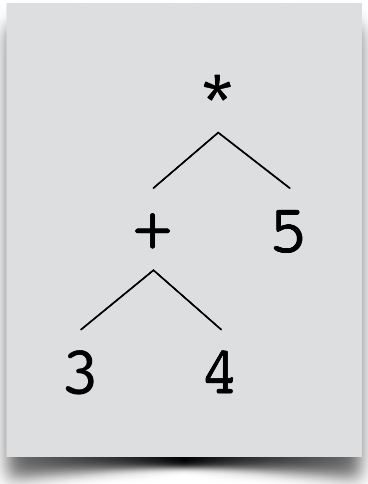


~~3 + 4 * 5~~

~~→ 7 * 5~~

~~→ 35~~

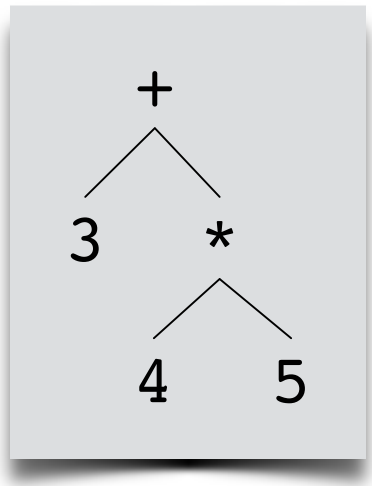




3 + 4 * 5

➡ 3 + 20

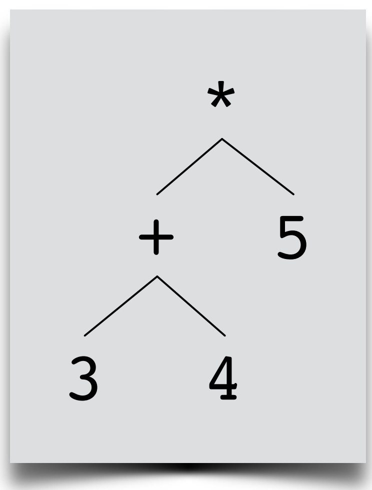
➡ 23



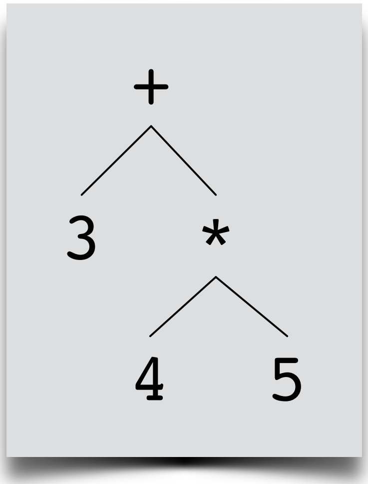
~~3 + 4 * 5~~

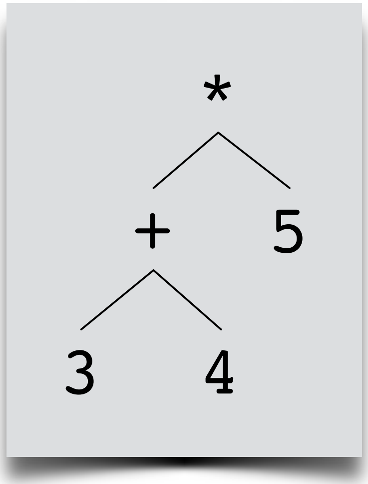
~~➡ 7 * 5~~

~~➡ 35~~



abstract syntax

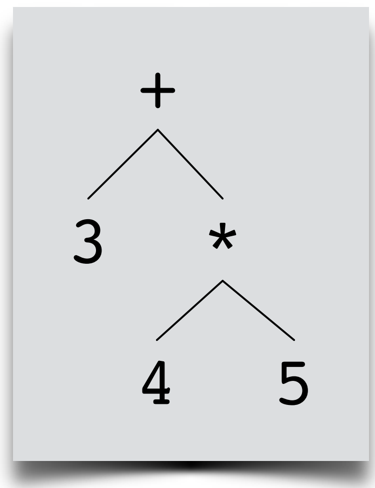




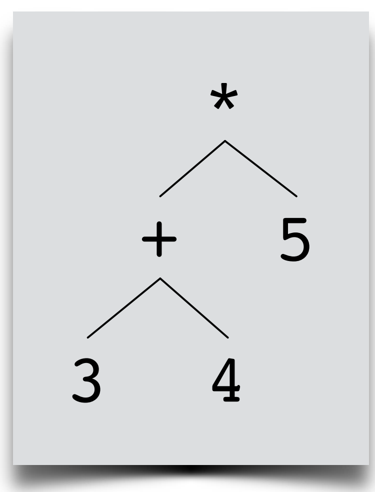
concrete syntax

abstract syntax

`3 + 4 * 5`



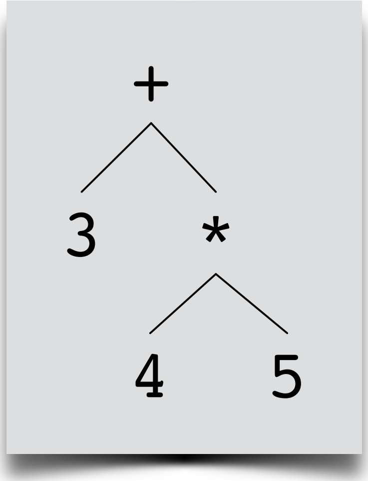
`(3 + 4) * 5`



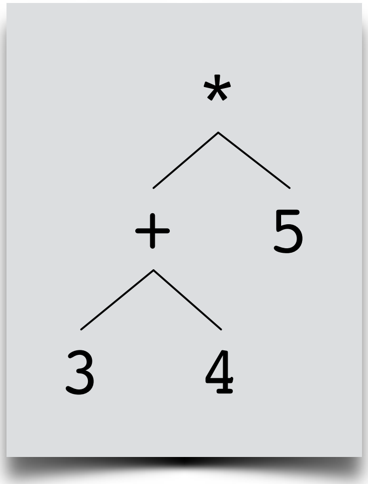
concrete syntax

abstract syntax

```
# 3 + 4 * 5 ;;
- : int = 23
```



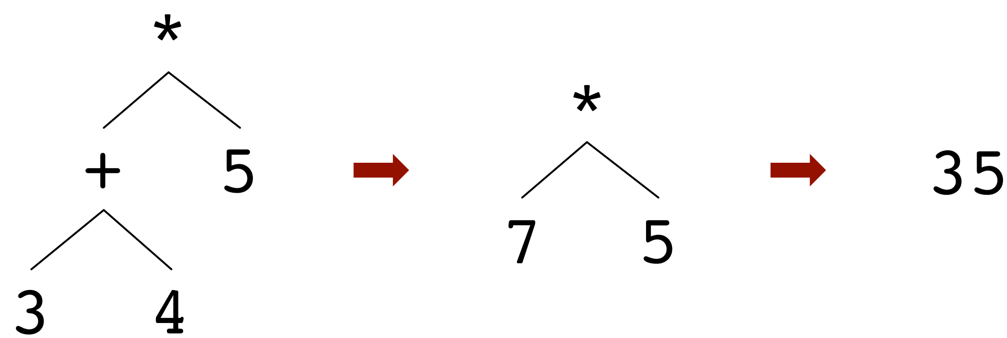
```
# (3 + 4) * 5 ;;
- : int = 35
```



value:

value: a result determined by calculation or measurement

`35`	*an integer value*
`true`	*a boolean value*
`fun x -> x + 1`	*a function value*




```
# (3 + 4) * 5 ;;
- : int = 35
```

type:

type: a set of things having traits or characteristics in common that distinguish them as a group


GOLD HILL
EST. — 1859
ELEV. — 8463
POP. — 118
TOTAL 10440

```c
#include <stdio.h>
#include <stdbool.h>

int main() {
  int name = "Gold Hill";
  int est = 1859;
  bool us = true;
  int sum = name + est + us;
  printf("Value : %d\n", sum);
}
```


find this code in: `joke.c`

OCaml is a *typed* language

Expressions have *types*

Statically typed:

- type of an expression can be determined just by looking at the code

Strongly typed:

- interpreter enforces type abstraction
- cannot use an integer as a record, function, string, etc.

Implicitly typed:

- interpreter can determine the types of most expressions on its own

```
# 3 + 4 * 5 ;;
- : int = 23
```

```
# 3 + 4 * 5 ;;
- : int = 23
# "three" + 4 * 5 ;;
Error: This expression has type string
but an expression was expected of type
int
```

```
# 3 + 4 * 5 ;;
- : int = 23
# "three" + 4 * 5 ;;
Error: This expression has type string
but an expression was expected of type
int
# 3.0 + 4.0 ;;
Error: This expression has type float but
an expression was expected of type int
```

```
# 3 + 4 * 5 ;;
- : int = 23
# "three" + 4 * 5 ;;
Error: This expression has type string
but an expression was expected of type
int
# 3.0 + 4.0 ;;
Error: This expression has type float but
an expression was expected of type int
# (+) ;;
- : int -> int -> int = <fun>
```

```
# 3 + 4 * 5 ;;
- : int = 23
# "three" + 4 * 5 ;;
Error: This expression has type string
but an expression was expected of type
int
# 3.0 + 4.0 ;;
Error: This expression has type float but
an expression was expected of type int
# (+) ;;
- : int -> int -> int = <fun>
# 3.0 +. 4.0 ;;
- : float = 7.
```

function:

function: a determinate mapping from one or more inputs to an output

```
fun x -> 2 * x
```

```
fun x -> 2 * x
```

anonymous function

```
fun x -> 2 * x
```

introduces a variable
naming the argument

```
fun x -> 2 * x
```

uses the argument by invoking its name

```
fun x -> 2 * x

fun x -> if x < 3 then x else x * x
```

Function application

In C:

- `f(3)`
- `f(3, 4, 5)`

8-7 Functions Defined by Equations

Objective To define a function by using equations.

Tickets to the senior class play cost \$5. Production expenses are \$500. The class's profit, p will depend on n, the number of tickets sold.

$$\text{profit} = \$5 \cdot (\text{number of tickets}) - \$500 \quad \text{or} \quad p = 5n - 500$$

The equation $p = 5n - 500$ describes a correspondence between the number of tickets sold and the profit. This correspondence is a function whose domain is the set of tickets that could possibly be sold.

$$\text{domain } D = \{0, 1, 2, \ldots\}.$$

The range is the set of profits that are possible, including "negative profits," or losses, if too few tickets are sold.

$$\text{range } R = \{-500, -495, -490, \ldots\}.$$

If we call this profit function P, we can use **arrow notation** and write the rule

$$P: n \rightarrow 5n - 500,$$

which is read "the function P that assigns $5n - 500$ to n" or "the function P that pairs n with $5n - 500$." We could also use **functional notation:**

$$P(n) = 5n - 500$$

which is read "P of n equals $5n - 500$" or "the value of P at n is $5n - 500$."

To specify a function completely, you must describe the domain of the function as well as give the rule. The numbers assigned by the rule then form the range of the function.

Example 1 List the range of

$$g: x \rightarrow 4 + 3x - x^2$$

if the domain $D = \{-1, 0, 1, 2\}$.

Solution In $4 + 3x - x^2$ replace x with each member of D to find the members of the range R.

$\therefore R = \{0, 4, 6\}$ ***Answer***

x	$4 + 3x - x^2$
-1	$4 + 3(-1) - (-1)^2 = 0$
0	$4 + 3(0) - 0^2 = 4$
1	$4 + 3(1) - 1^2 = 6$
2	$4 + 3(2) - 2^2 = 6$

Note that the function g in Example 1 assigns the number 6 to both 1 and 2. In listing the range of g, however, you name 6 only once.

Members of the range of a function are called **values of the function.** In Example 1, the values of the function g are 0, 4, and 6. To indicate that the function g assigns to 2 the value 6, you write

$$g(2) = 6,$$

which is read "g of 2 equals 6" or "the value of g at 2 is 6." Note that $g(2)$ is *not* the product of g and 2. It names the number that g assigns to 2.

Brown, Dolciani, Sorgenfrey, and Cole, *Algebra: Structure and Method*, 2000, page 379.

the set of tickets that could possibly be sold.

$$\text{domain } D = \{0, 1, 2, \ldots\}.$$

The range is the set of profits that are possible, including "negative profits," or losses, if too few tickets are sold.

$$\text{range } R = \{-500, -495, -490, \ldots\}.$$

If we call this profit function P, we can use **arrow notation** and write the rule

$$P\colon n \rightarrow 5n - 500,$$

which is read "the function P that assigns $5n - 500$ to n" or "the function P that pairs n with $5n - 500$." We could also use **functional notation:**

$$P(n) = 5n - 500$$

which is read "P of n equals $5n - 500$" or "the value of P at n is $5n - 500$."

To specify a function completely, you must describe the domain of the function as well as give the rule. The numbers assigned by the rule then form the range of the function.

Example 1 List the range of

$$g\colon x \rightarrow 4 + 3x - x^2$$

if the domain $D = \{-1, 0, 1, 2\}$.

x	$4 + 3x - x^2$
-1	$4 + 3(-1) - ($
0	$4 + 3(0) - 0^2 =$

Solution In $4 + 3x - x^2$ replace x with each the set of tickets that could possibly be sold.

$$\text{domain } D = \{0, 1, 2, \ldots\}.$$

The range is the set of profits that are possible, including "negative profits," or losses, if too few tickets are sold.

$$\text{range } R = \{-500, -495, -490, \ldots\}.$$

If we call this profit function P, we can use **arrow notation** and write the rule

$$P: n \rightarrow 5n - 500,$$

which is read "the function P that assigns $5n - 500$ to n" or "the function P that pairs n with $5n - 500$." We could also use **functional notation:**

$$P(n) = 5n - 500$$

which is read "P of n equals $5n - 500$" or "the value of P at n is $5n - 500$."

To specify a function completely, you must describe the domain of the function as well as give the rule. The numbers assigned by the rule then form the range of the function.

Example 1 List the range of

$$g: x \rightarrow 4 + 3x - x^2$$

if the domain $D = \{-1, 0, 1, 2\}$.

x	$4 + 3x - x^2$
-1	$4 + 3(-1) - ($
0	$4 + 3(0) - 0^2 =$

Solution In $4 + 3x - x^2$ replace x with each

Example 1 List the range of

$$g\colon x \to 4 + 3x - x^2$$

if the domain $D = \{-1, 0, 1, 2\}$.

x	$4 + 3x - x^2$
-1	$4 + 3(-1) - (-$
0	$4 + 3(0) - 0^2 =$
1	$4 + 3(1) - 1^2 =$
2	$4 + 3(2) - 2^2 =$

Solution In $4 + 3x - x^2$ replace x with each member of D to find the members of the range R.

$\therefore R = \{0, 4, 6\}$ ***Answer***

Note that the function g in Example 1 assigns the number 6 to both 1 and 2. In listing the range of g, however, you name 6 only once.

Members of the range of a function are called **values of the function.** In Example 1, the values of the function g are 0, 4, and 6. To indicate that the function g assigns to 2 the value 6, you write

$$g(2) = 6,$$

which is read "g of 2 equals 6" or "the value of g at 2 is 6." Note that $g(2)$ is *not* the product of g and 2. It names the number that g assigns to 2.

quoties Px vel Pa fuerit functio ipſarum a et x nullius dimenſionis. Deinde etiam obſeruaui, quoties in P litterae a et x eundem tantum vbique conſtituant dimenſionum numerum, toties Q ab integratione ipſius Pdx pendere. Ex quo, cum tam eximia conſequantur ſubſidia ad aequationes modulares inueniendas, maxime iuuabit inueſtigare, num forte aliae dentur huiusmodi functiones ipſius P, quae iisdem praerogatiuis gaudeant. Has igitur a priore inueſtigare conſtitui, quo ſimul methodus tales functiones inueniendi aperiatur.

§. 6. Si P eſt functio ipſarum a et x dimenſionum -1, ſeu z functio ipſarum a et x nullius dimenſionis, oſtendi fore $Px+Qa=o$, ſeu $Q=-\frac{Px}{a}$. Sumamus igitur eſſe $Q=-\frac{Px}{a}$ et quaeramus, qualis ſit P functio ipſarum a et x. At ſi $Q=-\frac{Px}{a}$ erit $dz=Pdx-\frac{Pxda}{a}$. Quamobrem P talis eſſe debebit functio ipſarum a et x, vt $dx-\frac{xda}{a}$ per eam multiplicatum euadat integrabile. Hic autem per integrabile non ſolum intelligo, quod integratione ad quantitatem algebraicam, ſed etiam quod ad quadraturam quamcunque reducitur. Si igitur generaliter inuenerimus quantitatem, in quam $dx-\frac{xda}{a}$ ductum ſit integrabile, ea erit quaeſitus valor ipſius P, eius proprietatis, vt ſit $Q=-\frac{Px}{a}$.

§. 7. Fit autem $dx-\frac{xda}{a}$ integrabile ſi multiplicatur per $\frac{1}{a}$, integrale enim erit $\frac{x}{a}+c$, deſignante c quantitatem conſtantem quamcunque ab a non pendentem. Quocirca, ſi $f(\frac{x}{a}+c)$ denotet functionem quamcunque ipſius $\frac{x}{a}+c$, fiet quoque $dx-\frac{xda}{a}$ integrabile, ſi multiplicetur per $\frac{1}{a}f(\frac{x}{a}+c)$. Qui valor cum ſit maxime generalis, erit $P=\frac{1}{a}f(\frac{x}{a}+c)$, et $Q=-\frac{Px}{a}$. Eſt vero $f(\frac{x}{a}+c)$ functio quaecunque ipſarum a et x nullius demenſionis. Quamobrem quoties Pa fuerit functio nullius dimenſionis ipſarum a et x, toties erit $Q=-\frac{Px}{a}$, ideoque aequatio modularis $dz=Pdx-\frac{Pxda}{a}$.

§. 8. Sit $Q=A-\frac{Px}{a}$, et A functio quaecunque ipſius a et conſtantium; erit $dz=Pdx+Ada-\frac{Pxda}{a}$ ſeu $dz-Ada=Pdx-\frac{Pxda}{a}$. In qua aequatione cum $dz-Ada$ ſit integrabile, debebit $Pdx-\frac{Pxda}{a}$ quoque eſſe integrabile. Hoc autem per praecedentem operationem euenit ſi $P=\frac{1}{a}f(\frac{x}{a}+c)$. Tum igitur erit $Q=A-\frac{x}{a^2}f(\frac{x}{a}+c)$. Simili ratione intelligitur ſi fuerit $P=X+\frac{1}{a}f(\frac{x}{a}+c)$, denotante X functionem ipſius x tantum, fore $Q=A-\frac{x}{a^2}f(\frac{x}{a}+c)$, vbi vt ante $f(\frac{x}{a}+c)$ exprimit functionem quamcunque ipſarum a et x nullius dimenſionis.

§. 9. Sit $Q=-\frac{nPx}{a}$, vbi n indicet numerum quemcunque; erit $dz=Pdx-\frac{nPxda}{a}$. Debebit ergo P talis eſſe quantitas, quae $dx-\frac{nxda}{a}$ ſi in id multiplicetur, reddat integrabile. Fit autem $dx-\frac{nxda}{a}$ integrabile, ſi ducatur in $\frac{1}{a^n}$, integrale enim erit $\frac{x}{a^n}$. Quare generaliter erit $P=\frac{1}{a^n}f(\frac{x}{a^n}+c)$. Atque quoties P talem

$-\frac{}{a}$. Quamobrem P talis esse debebit functio ipsa-
rum a et x, vt $dx - \frac{xda}{a}$ per eam multiplicatum euadat integrabile. Hic autem per integrabile non ſolum intelligo, quod integratione ad quantitatem algebraicam, ſed etiam quod ad quadraturam quamcunque reducitur. Si igitur generaliter inuenerimus quantitatem, in quam $dx - \frac{xda}{a}$ ductum fit integrabile, ea erit quaeſitus valor ipſius P, eius proprietatis, vt ſit $Q = -\frac{Px}{a}$.

§. 7. Fit autem $dx - \frac{xda}{a}$ integrabile ſi multiplicatur per $\frac{1}{a}$, integrale enim erit $\frac{x}{a} + c$, deſignante c quantitatem conſtantem quamcunque ab a non pendentem. Quocirca, ſi $f(\frac{x}{a} + c)$ denotet functionem quamcunque

Function application

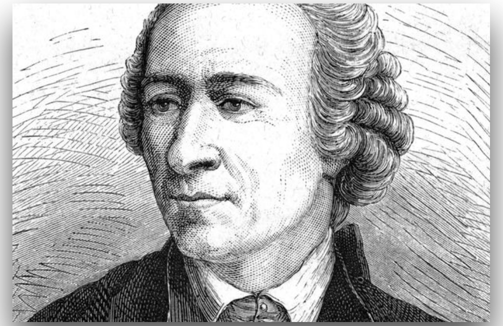
In C:

- `f(3)`
- `f(3, 4, 5)`

Function application

In C:

- `f(3)`
- `f(3, 4, 5)`

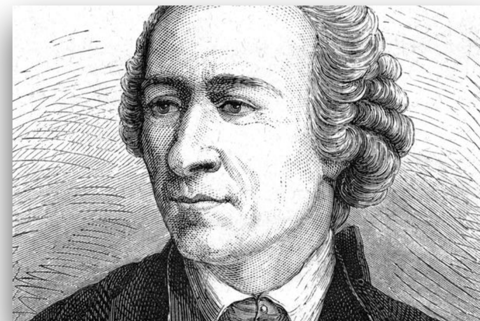


Leonhard Euler

Function application

In C:

- `f(3)`
- `f(3, 4, 5)`



Leonhard Euler

Alonzo Church

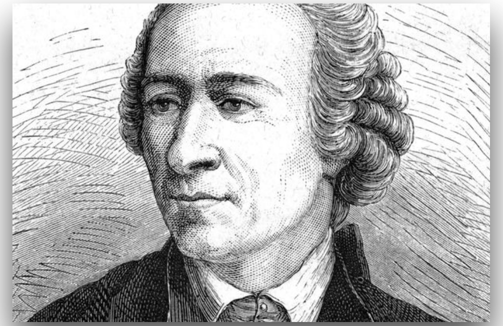
Function application

In C:

- `f(3)`
- `f(3, 4, 5)`

In OCaml:

- `f 3`



Leonhard Euler

Alonzo Church

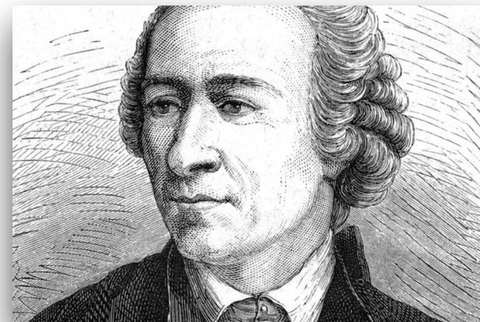
Function application

In C:

- `f(3)`
- `f(3, 4, 5)`

In OCaml:

- `f 3`
- `((f 3) 4) 5`



Leonhard Euler

Alonzo Church

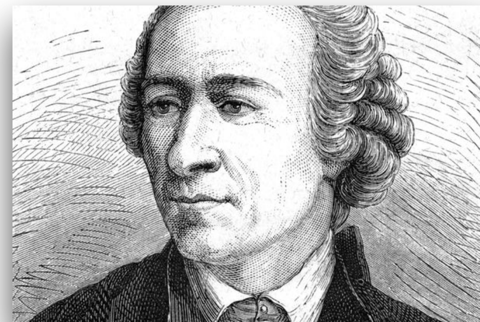
Function application

In C:

- `f(3)`
- `f(3, 4, 5)`

In OCaml:

- `f 3`
- `((f 3) 4) 5`
- `f 3 4 5`



Leonhard Euler

Alonzo Church

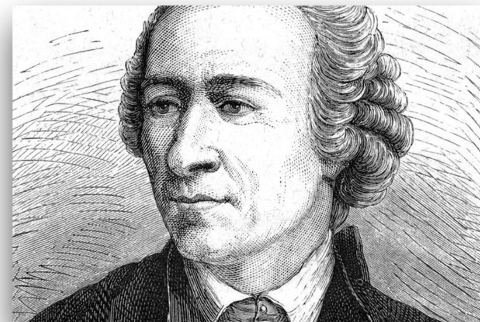
Function application

In C:

- `f(3)`
- `f(3, 4, 5)`

In OCaml:

- `f 3`
- `((f 3) 4) 5`
- `f 3 4 5`
- `f(3, 4, 5)`



Leonhard Euler

Alonzo Church

Semantics of function application

```
   (fun x -> x / 2) (3 * 5)
➡  (fun x -> x / 2) 15
➡  15 / 2
➡  7
```

Semantics of function application

```
(fun x -> x / 2) (3 * 5)
➡ (fun x -> x / 2) 15
➡ 15 / 2
➡ 7
```

evaluate argument to value

Semantics of function application

```
(fun x -> x / 2) (3 * 5)
➡ (fun x -> x / 2) 15
➡ 15 / 2
➡ 7
```

substitute function's argument variable with value being applied to

definition:

definition: the act of stating a precise meaning

```
let x = 3 * 5 in
x * x
```

local naming

```
let x = 3 * 5 in
x * x
```

```
let x = 3 * 5 in
x * x
```

```
➡ let x = 15 in
x * x
```

evaluate definiendum
(3 * 5) to a value

```
let x = 3 * 5 in
x * x
```

➡

```
let x = 15 in
x * x
```

name is only available in the *body* of the `let`, its *scope*

```
let x = 3 * 5 in
x * x
```

➡
```
let x = 15 in
x * x
```

➡
```
15 * 15
```

substitute definiens (`x`) in body

```
   let x = 3 * 5 in
   x * x

➡  let x = 15 in
   x * x

➡  15 * 15

➡  225
```

substitute definiens (x) in body

```
let x = 3 in
let x = x * 2 in
x + 1
```

```
let x = 3 in
let x = x * 2 in
x + 1
```

```
let x = 3 in
let x = x * 2 in
x + 1
```

```
let x = 3 in
let x = x * 2 in
x + 1
```

➡
```
let x = x * 2 in
x + 1
```

```
let x = 3 in
let x = x * 2 in
x + 1
```

➡
```
let x = x * 2 in
x + 1
```

➡
```
let x = (x * 2) * 2 in
x + 1
```

➡

```
let x = 3 in
let x = x * 2 in
x + 1
```

```
let x = 3 in
let x = x * 2 in
x + 1

➡ let x = 3 * 2 in
  x + 1

➡ let x = 6 in
  x + 1

➡ 6 + 1

➡ 7
```

```
let x = 3 in
let x = x * 2 in
x + 1

➡ let x = 3 * 2 in
  x + 1

➡ let x = 6 in
  x + 1

➡ 6 + 1

➡ 7
```

Names in the definiendum refer *outside* the scope of the definition

```
let half = fun x -> x / 2 in
half (3 * 5)

➡ (fun x -> x / 2) (3 * 5)

➡ (fun x -> x / 2) 15

➡ 15 / 2

➡ 7
```

functions are values too

```
let half = fun x -> x / 2 in
half (3 * 5)
```

➡ `(fun x -> x / 2) (3 * 5)`

➡ `(fun x -> x / 2) 15`

➡ `15 / 2`

➡ `7`

```
let half = fun x -> x / 2 in
half (3 * 5)

➡ (fun x -> x / 2) (3 * 5)

➡ (fun x -> x / 2) 15

➡ 15 / 2

➡ 7
```

evaluate argument to a value

```
let half = fun x -> x / 2 in
half (3 * 5)

➡ (fun x -> x / 2) (3 * 5)

➡ (fun x -> x / 2) 15

➡ 15 / 2

➡ 7
```

substitute function argument with value being applied to

```
let double = fun x -> 2 * x in
double 5
```

```
let double = fun x -> 2 * x in
double 5
```

local naming of
a function

```
let double = fun x -> 2 * x in
double 5

let double x = 2 * x in
double 5
```

```
let double = fun x -> 2 * x in
double 5
```

```
let double x = 2 * x in
double 5
```

"syntactic sugar":
alternative concrete syntax for
programming convenience

abstraction:

abstraction: the process of viewing a set of apparently dissimilar things as instantiating an underlying identity

Higher-order functions and functional programming

Polymorphism and generic programming

Handling anomalous conditions

Algebraic data types

Abstract data types and modular programming

Mutable state and imperative programming

Loops and procedural programming

Infinite data structures and lazy programming

Decomposition and object-oriented programming

For next time (lab 1)...

Read chapters 1–6 (book.cs51.io)

Read "On doing well in CS51"

Work on Problem Set 0 (installing the required course software), due Monday 11:59pm

- Office hours to help you get things installed listed in CS51 Canvas calendar

Euclid of Alexandria

cs51.io