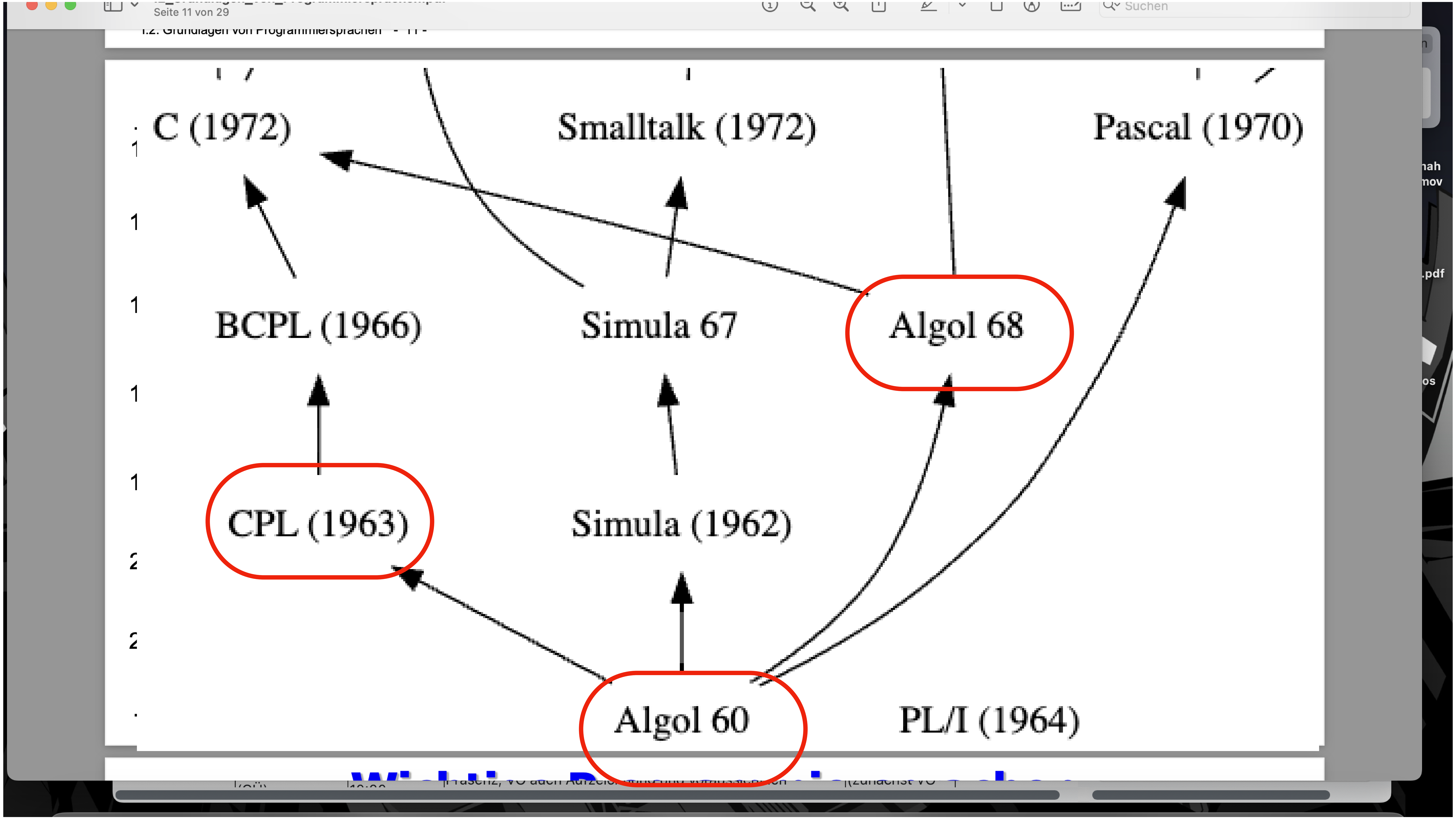
Historic Programming Languages

Fortgeschrittene Programmierkonzepte

Sandro, Toygun, Tufan

Konzept für ein Anfang folgt

Intro fehlt



Seite 11 von 29

1.2. Grundlagen von Programmiersprachen - 11 -

↳ keine Bilder einfach kopieren

FORTRAN

FORTRAN (FORmula TRANslation)

FORTRAN

Fortran

Typescript

Java

C#

Python

C

?

FORTRAN

Fortran

is this much detail necessarry ?

- First version from the 1950s
- Multiple newer versions like Fortran 77, 90, 95 or 2003
- Latest version in 2018
- Next version will be published this year
- ~~Files are saved as *.f77, *.f90 etc. or as *.f~~

Invention

Why was it developed?

- It was the first High-level programming language
- Therefore it was the first, that needed an Compiler
- Fortran is a procedural, structured programming language
- It should make programming much less difficult
- Developed by John W. Backus

Innovations

Fortran featured many paradigms, which are still found today

- Variables and Constants with different data types
- Control structures
- Subroutines
- Operators
- (Object oriented programming)

Applications

Where can we still find Fortran in today‘s world?

- Weather prediction
- Medicine
- Computational fluid dynamics
- Computational chemistry and physics
- benchmarking the world‘s fastest supercomputers

FORTRAN 90

Syntax

Fortran 90

Syntax

```
program helloworld
    !I am a comment
    print *, "Hello World!"
end program helloworld
The compiler ignores me.
```

- Every program has to be inside the program/end program block.
- The rest is going to be ignored by the compiler
- Line-comments are made with „!“
- Output to the console is made with the print command and an asterisk as the first parameter
- Capitalization is irrelevant

Fortran 90

Syntax

```
pRoGRAm helloworld
    !I am a comment
    PRINT *, "Hello World!"
end pRoGRAm helloworld
The compiler ignores me.
```

moving

- Every program has to be inside the program/end program block.
- The rest is going to be ignored by the compiler
- Line-comments are made with „!“
- Output to the console is made with the print command and an asterisk as the first parameter
- Capitalization is irrelevant

Fortran 90

Data Types

Java	Fortran
int	Integer
double	Real
boolean	Logical
char	Character
-	Complex

Fortran 90

Variables

```
program variables
    implicit none
    integer :: i, j
    real :: x
    character :: char
    logical :: flag
    complex :: z

    i = 10
    char = 'a'
    x = 3.14159
    flag = .true.
    z = (1.0, 2.0)

    print *, i, char, x, flag, z
end program variables
```

You have a lot of bullet-points and text

- Fortran supports automatic type detection
- variables must be declared with an „::“
- Java‘s initialization of variables is comparable to Fortrans
- By giving multiple arguments, Every variable is printed

Fortran 90

Variables

```
program variables
    implicit none
    character(len=4) :: name
    read *,name
    print *,'Hey, ',name,'!'
end program variables
```

- Strings are realized with characters

Fortran 90

Variables

```
program variables
    implicit none
    character(len=4) :: name
    read *,name
    print *,'Hey, ',name,'!'
end program variables
```

moves again

- Strings are realized with characters
- Rick → Hey, Rick!
- Johnny → Hey, John!
- Tim → Hey, Tim !

Fortran 90

Operators

Java	Fortran
<	.LT.
<=	.LE.
==	.EQ.
!=	.NE.
>=	.GE.
>	.GT.

420 .NE. 69 returns .true.

Python	Fortran
+	+
-	-
*	*
/	/
**	**
+ ? ↔	//

Fortran hugely impacted modern languages

Fortran 90

Operators

Python	Fortran
and	.AND.
or	.OR.
==	.EQV.
!=	.NEQV.
not	.NOT.

.true. .NEQV. .false. returns .true.

Fortran 90

Control structures

```
!Fortran:
if ( area .LE. paint) then
        print *, 'The area can be painted'
    else
        print *, 'The area can not be
painted'
    end if
```

```
//Java:
if (area <= paint) {
            System.out.println("The area can be
painted");
        } else {
            System.out.println("The area can
not be painted");
        }
```

- It is easy to see the similarities between Fortran and Java
- The program flow can be easily modified depending on values of variables

Fortran 90

Control structures - loops

use colour to distinguish from code

Fortran

```
integer :: i
do i=1,10
    print *, i
end do
```

Java

```
for(int i = 1; i <= 10; i++) {
    System.out.println(i);
}
```

Fortran

```
integer :: n = 2
do while (n .LE. 100)
    print *, n
    n = n ** 2
end do
```

Java

```
int n = 2;
while (n <= 100) {
    System.out.println(n);
    n = (int) Math.pow(n, 2);
}
```

Fortran 90

Paintingproblem

```
program painting
    implicit none
    real :: r, pi, area, paint
    parameter (pi = 3.141592653589793238462643383279502884197169399375105820974944592307816406286208998628034825342
    1170679821480865132823066470938446095505822317253594081284811174502841027019385211055596446229489549303819644288
    8109756659334461284756482337867831652712019091456485669234603486104543266482133936072602491412737245870066063155
    5881748815209209628292540917153643678925903600113305305488204665213841469519415116094330572703657595919530921861
    1173819326117931051185480744623799627495673518857527248912279381830119491298336733624406566430860213949463952247
    3719070217986094370277053921717629317675238467481846766940513200056812714526356082778577134275778960917363717872
    1468440901224953430146549585371050792279689258923542019956112129021960864034418159813629774771309960518707211
    349999998372978049951059731732816096318595024459455346908302642522308253344685035261042875546873115956286388235
    3787593751957781857780532171226806613001927876611195909216420198938095257201065485863278865936153381827968230301
    9520353018529689957736225994138912497217752834791315155748572424541506959508295331168617278558890750983817546
    3746493931925506040092770167113900984882401285836160356370766010471018194295559619894676783744944825537977472684
    7104047534646208046684259069491293313677028989152104752162056966024058038150193511253382430035587640247496473263
    9141992726042699227967823547816360093417216412199245863150302861829745557067498385054945885869269956909272107
    9750930295532116534498720275596023648066549911988183479775356636980742654252786255181841757467289097777279380008
    1647060016145249192173217214772350141441973568548161361157352552133475741849468438523323907394143334547762416862
    5189835694855620992192221842725502542568876717904946016534668049886272327917860857843838279679766814541009538
    8378636095068006422512520511739298489608412848862694560424196528502221066118630674427862203919494504712371378696
    0956364371917287467764657573962413890865832645995813390478027590099465764078951269468398352595709825822620522489
    4077267194782684826014769909026401363944374553050682034962524517493996514314298091906592509372216964615157098583
    8741059788595977297549893016175392846813826868386894277415599185592524595395943104997252468084598727364469584865
    3836736222626099124608051243884390451244136549762780797715691435997700129616089441694868555848406353422072225828
    4886481584560285060168427394522674676788952521385225499546667278239864565961163548862305774564980355936345681743
    241125150760694794510965960940252288797108931456691368672287489405601015033086179286809208747609178249385
```

Fortran 90

Paintingproblem

```fortran
program painting
    implicit none
    real :: r, pi, area, paint
    parameter (pi = 3.14159265358979323846264338327950)
    print *, 'Enter radius of circle'
    read *, r
    print *, 'Enter Area that can be covered in paint'
    read *, paint

    area = pi * r**2

    if ( area .LE. paint) then
        print *, 'The circle can be painted'
    else
        print *, 'The circle cannot be painted'
    end if
end program painting
```

Output:

```
Enter radius of circle
5
Enter Area that can be covered in paint
78
The circle cannot be painted!
```

Output:

```
Enter radius of circle
5
Enter Area that can be covered in paint
79
The circle can be painted
```

Fortran 90

Paintingproblem

```java
import java.util.Scanner;

public class Paintingproblem{

    public Paintingproblem() {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter the radius of the circle: ");
        double radius = scanner.nextDouble();
        System.out.println("Enter the amount of paint: ");
        double paint = scanner.nextDouble();
        double area = Math.PI * Math.pow(radius, 2);
```

```java
if (paint >= area) {
            System.out.println("The area can be painted");
        } else {
            System.out.println("The area can not be painted");
        }

scanner.close();
    }

    public static void main(String[] args) {
        new Paintingproblem();
    }
}
```

Fortran 90

Arrays

```
program arrays
    implicit none
    real, dimension(4) :: x
    x = (/ 3.141, 2.718, -10.01, 999.9 /)
    print *, x(1)
    print *, x(5)
end program arrays
```

(4,4)

Index	Value
1	3.141
2	2.718
3	-10.01
4	999.9

```
output1: 1.14100003
output2: 2.85741012E-37
```

⚠

Fortran 90

Functions and Subroutines

```
program function
    real :: my_square
    my_square = square(4.0)
    print *, my_square
end program function

function square(n)
    implicit none
    real :: square
    real :: n

    square = n**2
end function square
```

- intrinsic functions
  - MIN(5, -2, 64, 0) → -2
  - MAX(6.4, 18.0, -1.5, 9.99) → -1.5

Fortran 90

Functions and Subroutines

```
program subroutines
    implicit none
    integer :: age = 20
    call print_age(age)
end program subroutines

subroutine print_age(age)
    integer :: age
    print *, "I am ", age, " years
old."
end subroutine print_age
```

ALGOL (algorithmic language)

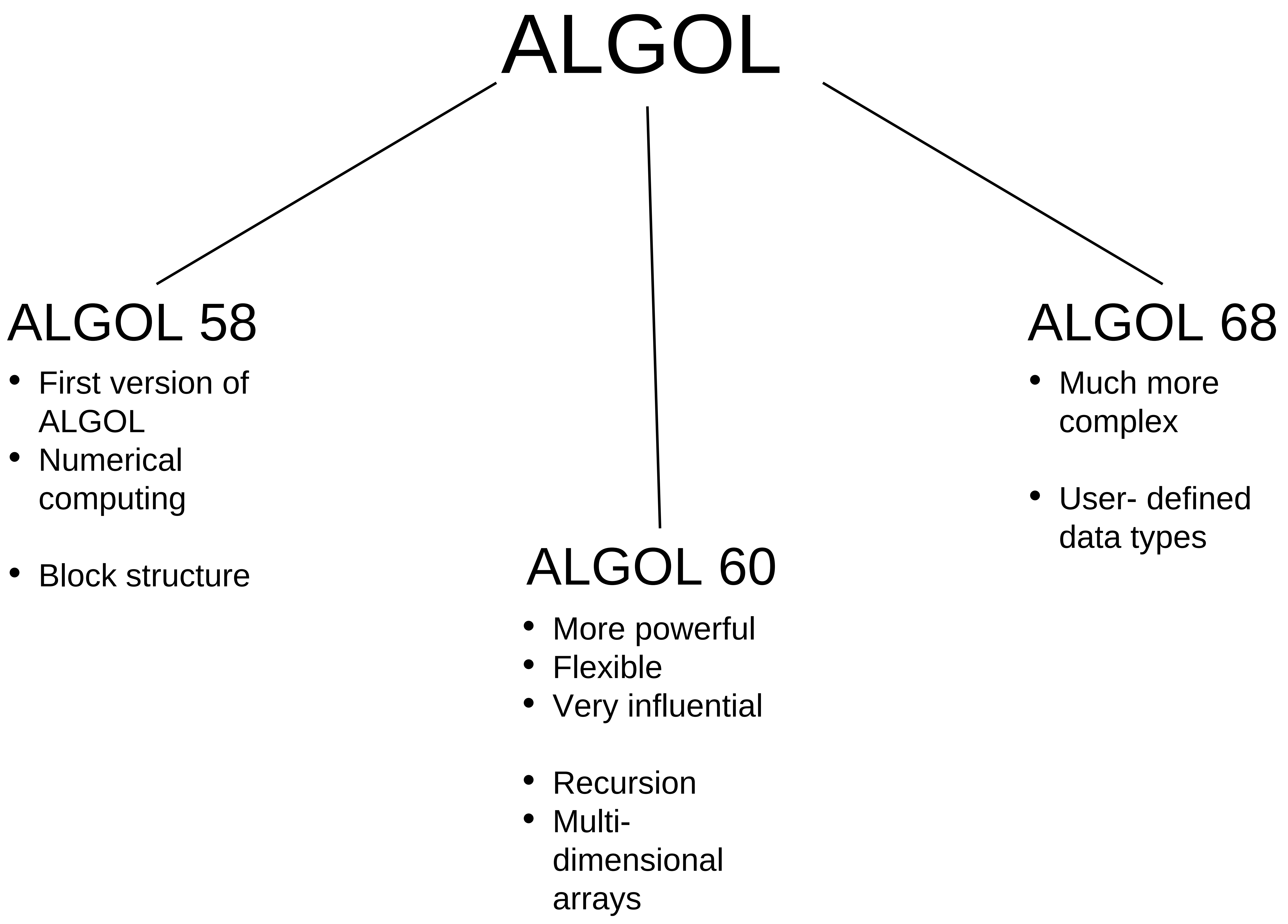
Was noch work in progress ist

Sorry

- Einleitung in das Thema ist nur ein Konzept bis jetzt
- Der History/Innovations/Area of application Teil ist unfertig/ausbaufähig
- Kleinigkeiten und Feinheiten

ALGOL

What’s that?



ALGOL

ALGOL 58

- First version of ALGOL
- Numerical computing

- Block structure

ALGOL 60

- More powerful
- Flexible
- Very influential

- Recursion
- Multi-dimensional arrays

ALGOL 68

- Much more complex

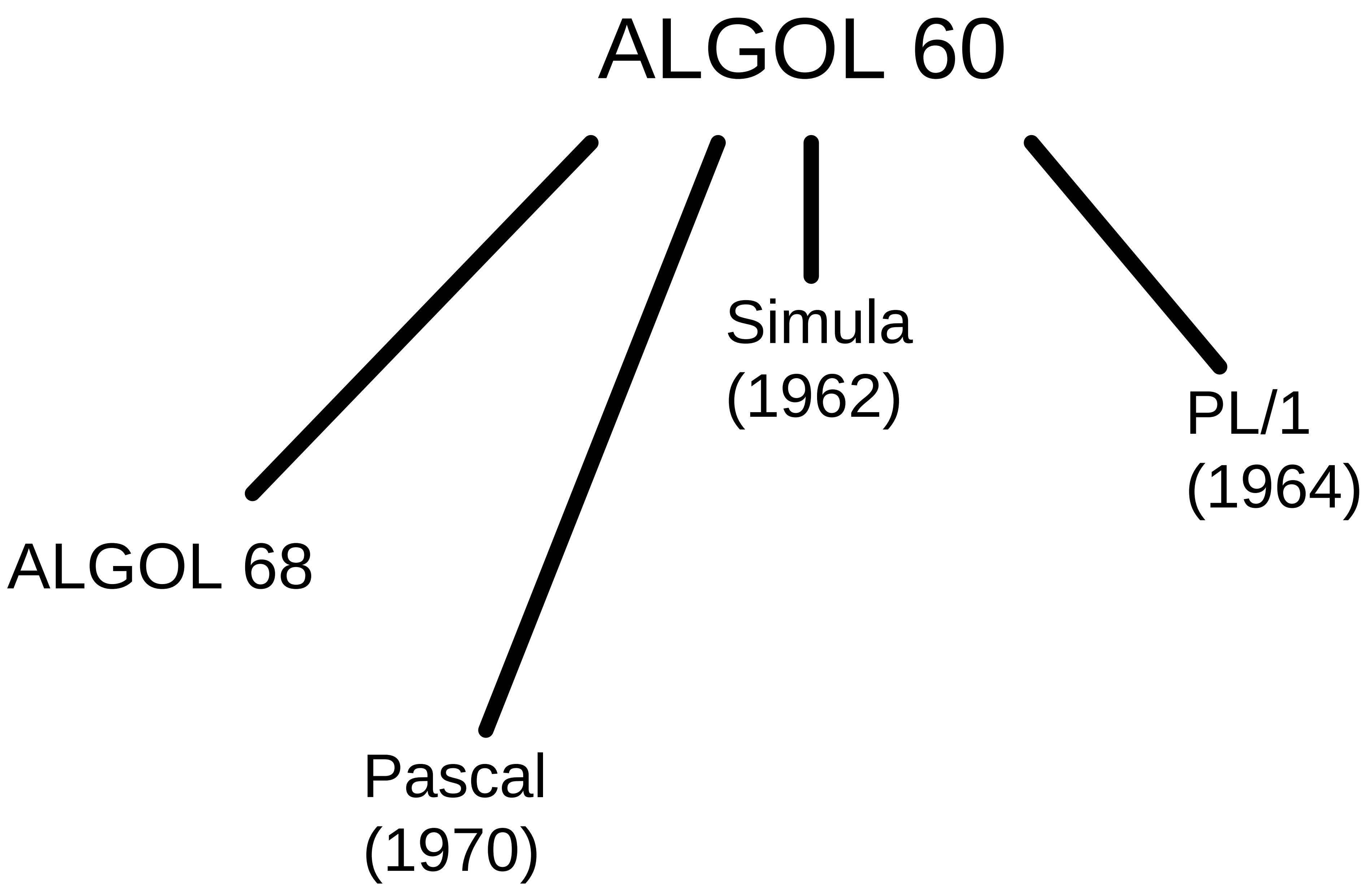
- User- defined data types

ALGOL 60

ALGOL 60

History

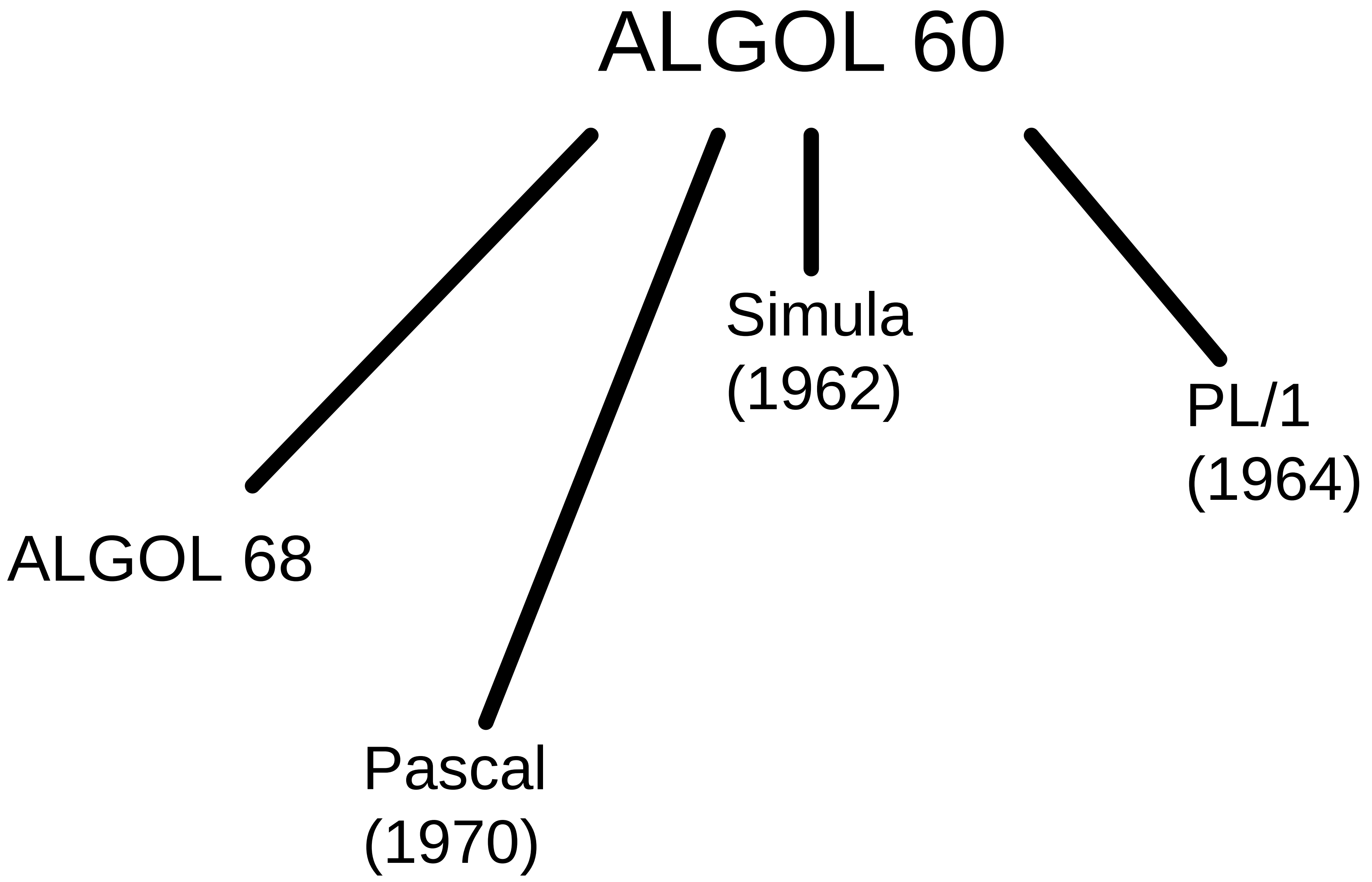
- Developed by an international comitee
- Designed for mathematical computing in mind
- Expressive and powerful



ALGOL 60

Innovations

- Recursion
- Lexical scoping
- Nested block structures
- Semicolons as statement terminators
- Clearer syntax, better to learn


ALGOL 60

ALGOL 68

Pascal (1970)

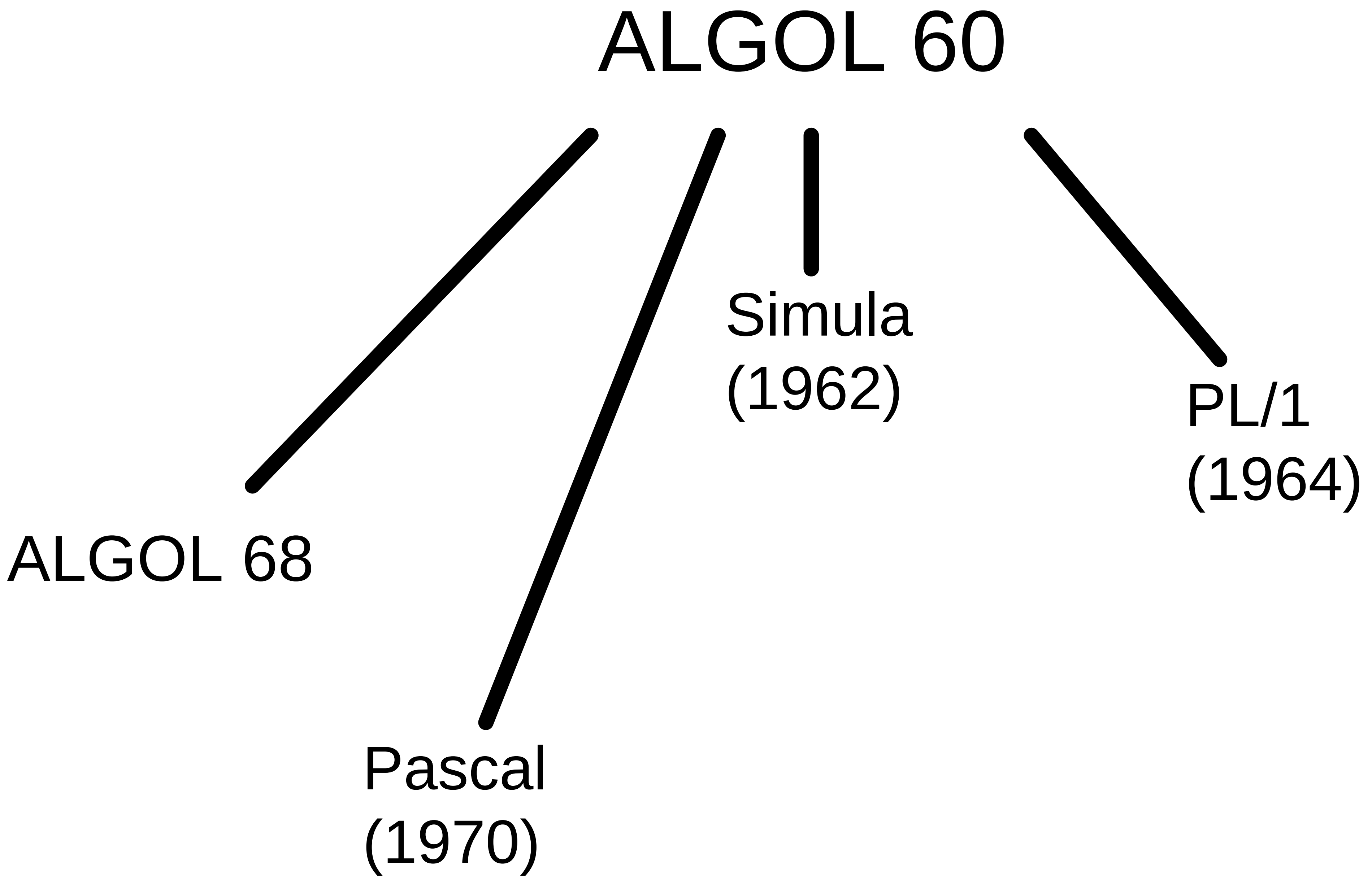
Simula (1962)

PL/1 (1964)

ALGOL 60

Areas of application

- Algorithm development
  - Clear and concise
- Scientific and engineering calculations



Syntax

Use same code style as before (maybe a bit small)

Code blocks

Syntax

- Code blocks defined with *begin/end*
- *Data types: integer, real, boolean*
- *Operators:* +, -, *, /, ÷, ↑

```
begin
    integer A, X;    real Z;    boolean Z;
    Z := 2.5;
    A := 5;
    X := 8;
     outinteger(1, A + X);
     outinteger(1, A * X);
     outinteger(1, Z / A);
     outinteger(1, X ÷ A);
     outinteger(1, X ↑ A);
end
```

```
Output: 8   40   0.5   1   32768
```

Lexical Scoping

Syntax

- Scope of variable determined by position in code/which block
    - Can be accessed within block and any block nested within
- Incorporated in Java, Python, JavaScript

```
begin
   integer A, X;
   A := 5;
   X := 8;                    comment outer X;
    begin
        integer X, Y;
        X := 5;               comment inner X assigned here;
        Y := 10;
    end
  outinteger(1,X);
 Y := 12;                comment error! Y not defined in outer scope;
 end
```

Output: 8

Arrays

Syntax

- Every array declared with one data type
    - real array by default
- array[*start*:*end*] ≠ Java

```
begin
    procedure arrayproc(n); value n; integer n;
   begin
      integer array x[0:n-1];
     x[0]:=10;
     x[1]:=11;
     x[2]:=12;
     x[3]:=13;

      outstring(1,“Value at index 2: ");
     outinteger(1,x[2]);
   end
 integer n := 4;

 arrayproc(n)
end
```

```
Output: Value at index 2: 12
```

Control structures

Syntax

If-then-else

```
integer i, j;
 i := 1;
 j := 8;
   IF i=1 THEN outinteger(1,“I“);

    IF i<j THEN outstring(1,“I<j“)
            ELSE outstring(1,“i>=j“);
```

```
Output: 1 i<j
```

For loops

```
integer i, j;
 FOR i:=1 STEP 1 UNTIL 5 DO
begin
      FOR J:=1 STEP 1 UNTIL i DO
         outstring(1,“ * ");
end
```

```
Output:  *
         **
         ***
         ****
         *****
```

Running example

Example

Circle problem

```
begin
  procedure circleAreaProblem(radius,area); real area; radius;
real pi := 3.14159;
   begin
       real circleArea := pi * radius * radius;
       if circleArea <= area then
        outstring(1, "The circle can be painted“)
     else
        outstring(1, "The circle cannot be painted“)
   end
 circleAreaProblem(5, 75);
 circleAreaProblem(5, 79);
end
```

```
Output: The circle cannot be painted

                The circle can be painted
```

What did ALGOL do 60 good?

Goods

ALGOL 60

- It has recursion
- Block structure
- Lexical scoping
- Clean and consistent syntax

→ Used in modern languages like Java

Bads

ALGOL 60

- No build in I/O facilities → Lack of standardization
- Limited set of data types → Hard to write with complex data structures
- No object-oriented programming → Less suited for large scale programms

Conclusion

Conclusion

ALGOL 60

- Legacy Language

Goods

- It has recursion
- Block structure
- Lexical scoping
- Clean and consistent syntax

Bads

- No build in I/O facilities
- Limited set of data types
- No object-oriented programming

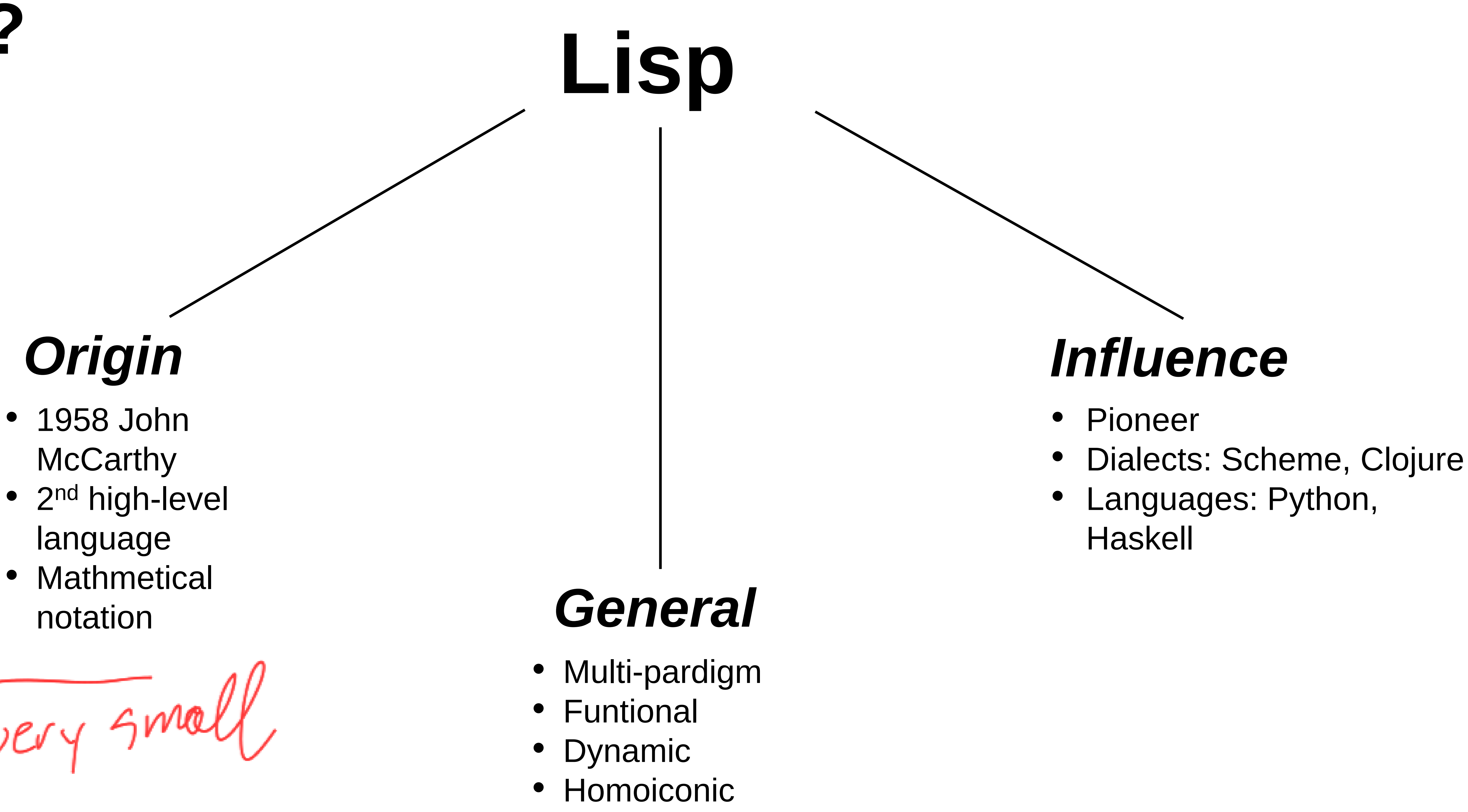
Lisp (List Processing)

TODOS and whats missing in the Lisp section

- Innovations/Application/Conclusion slides need improvement
- Comparison with Haskell not 100% finished

Lisp

What is Lisp?



Lisp

Origin

- 1958 John McCarthy
- 2nd high-level language
- Mathmetical notation

very small

General

- Multi-pardigm
- Funtional
- Dynamic
- Homoiconic

Influence

- Pioneer
- Dialects: Scheme, Clojure
- Languages: Python, Haskell

Innovations

Lisp

Innovations

- S-expressions
- Macros
- Garbage Collection
- Recursion
- Dynamic Typing
- Higher-order functions

Application

Application

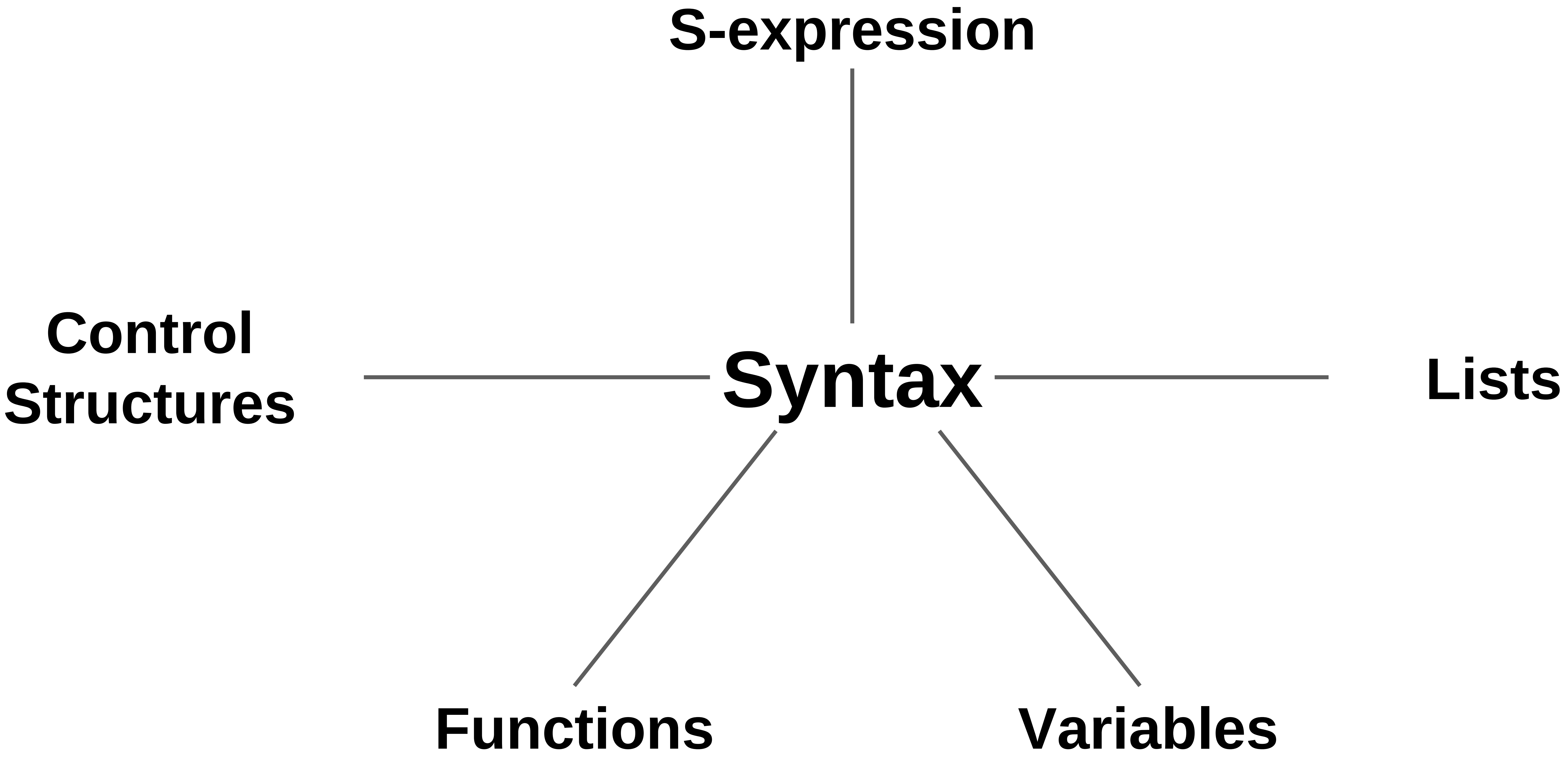
Lisp

- AI: Lisp-Machines had GUI and IDE
- Boeing and Airbus use Common Lisp
  - Software package Piano
- AutoCAD implemented in AutoLisp
- MTU Aero Engines applies AllegroCL
- Clojure in Wallmart data management system (+5000)

Syntax

Lisp

Syntax



S-expression

Control Structures

Syntax

Lists

Functions

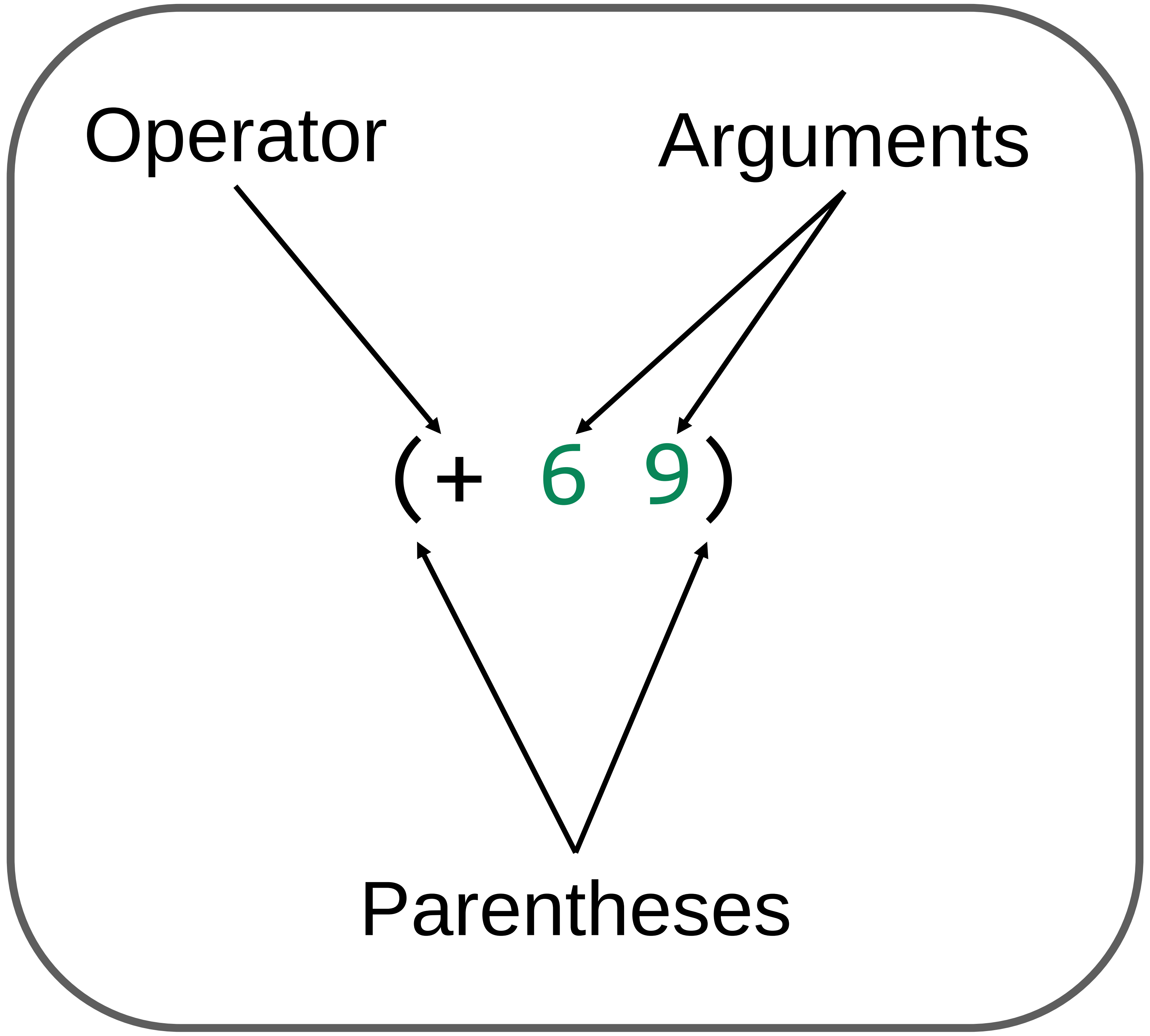
Variables

S-expressions

S-expressions

Syntax

- Lisp: “Lots of Irritating Superfluous Parentheses”
- Prefix Notation



Operator

Arguments

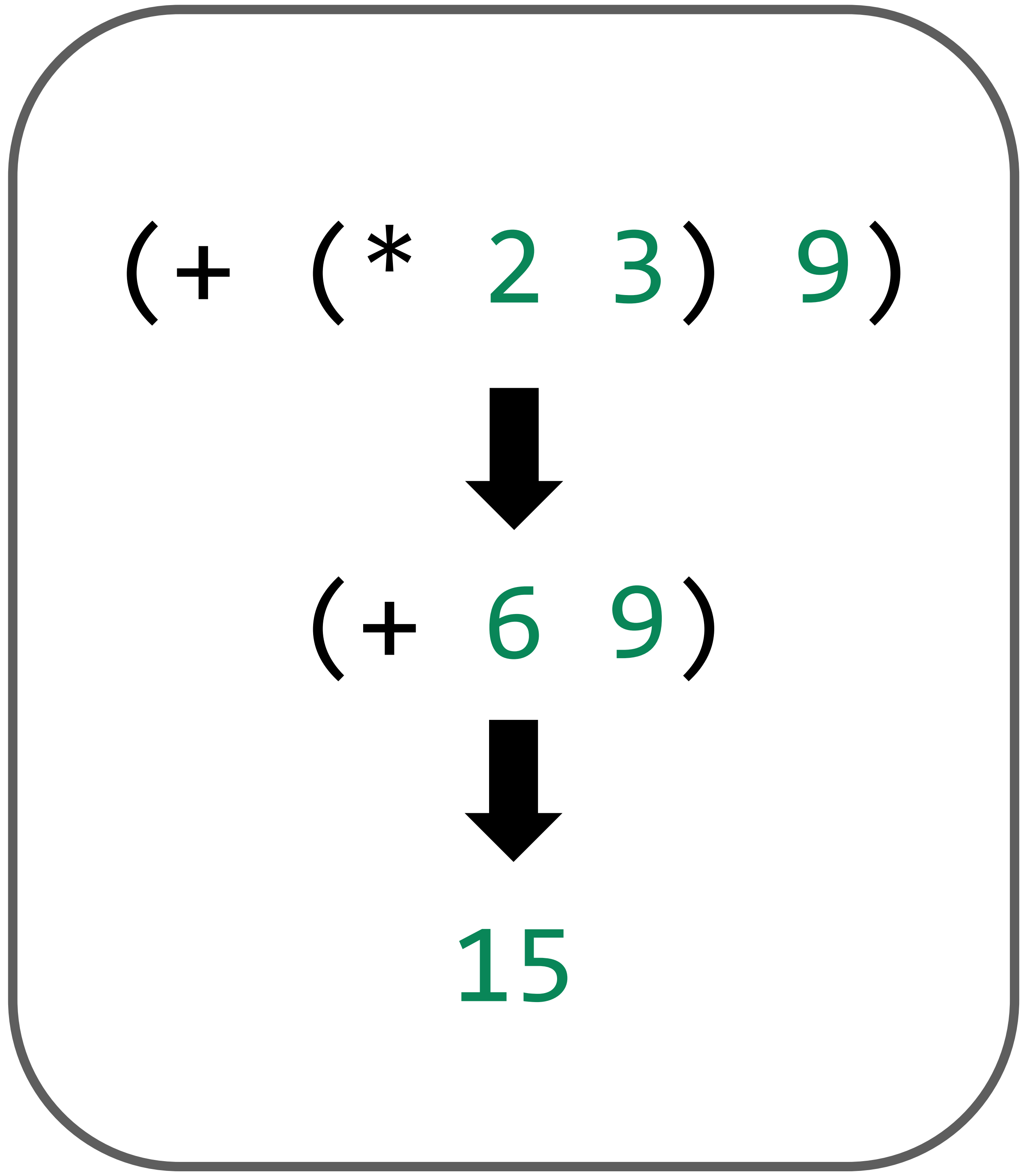
```
(+ 6 9)
```

Parentheses

S-expressions

Syntax

- Lisp: “Lots of Irritating Superfluous Parentheses”
- Prefix Notation
- Every expression gets evaluated



```
(+ (* 2 3) 9)
(+ 6 9)
15
```

S-expressions

Syntax

- Lisp: “Lots of Irritating Superfluous Parentheses”
- Prefix Notation
- Every expression gets evaluated
- Basic building blocks
  1. Atoms: numbers, string of numbers and characters
  2. Lists

Valid	Invalid
73842	(hello
name	666number
Number#21	Hi world

Lists

Lists

Syntax

Haskell

```
[]

[1,2,3]

1 : 2 : 3 : []
```

Lisp

```
nil

(1 2 3)

(cons 1 (cons 2 (cons 3 nil)))
```

Lists

Haskell

```
[]

[1,2,3]

1 : 2 : 3 : []
```

```
head [1,2,3] = 1

tail [1,2,3] = [2,3]
```

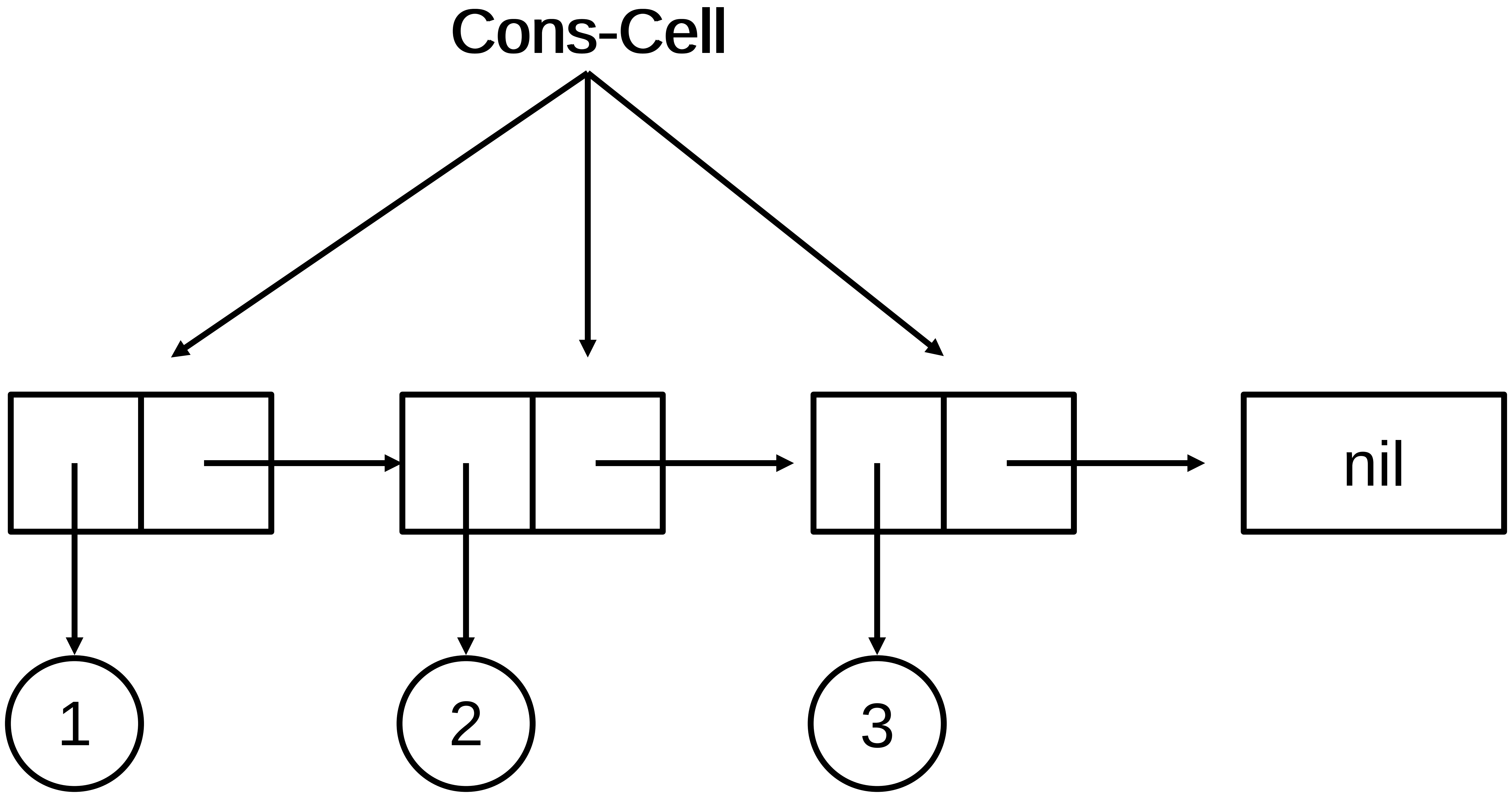
Lisp

```
nil

(1 2 3)

(cons 1 (cons 2 (cons 3 nil)))
```

```
(car '(1 2 3)) = 1

(cdr '(1 2 3)) = (2 3)
```

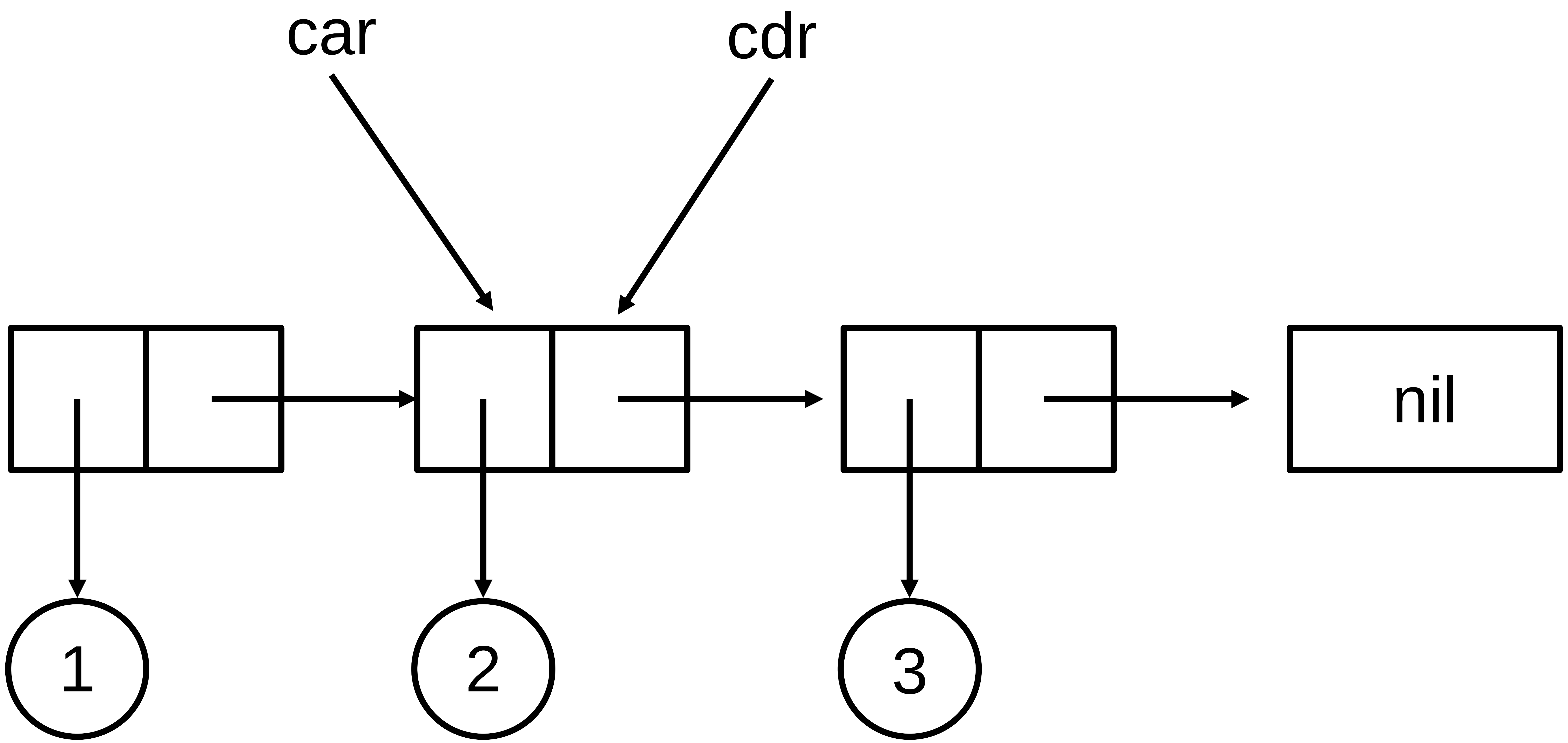
Lists

Syntax



Lists

Syntax



Variables

Variables

Syntax

- Declaration and initialization
- Only declaration
- Set/Change variable and local variable
- Local variables
- Constants

```
(defvar number#78 78)

(defvar number#12)

(setq number#12 12)

(let ((four 4) (five 5))
         (write (+ four five))

(defconstant giesl 1)
```

Code Example

Circle-Paint Program

- Get Input from user:
  - Radius of circle
  - How much paint
- Output if there is enough paint

Haskell

```
circle :: IO()
circle = do
  putStrLn "Enter radius of circle: "
  input1 <- getLine
  putStrLn "Enter how much paint you have: "
  input2 <- getLine
  let rad = (read input1)
  let paint = (read input2)
  let area =  (rad*rad*pi)
  if area <= paint then putStrLn "The circle CAN be painted!"
                   else putStrLn "The circle CAN NOT be painted!"
```

Code Example

Circle-Paint Program

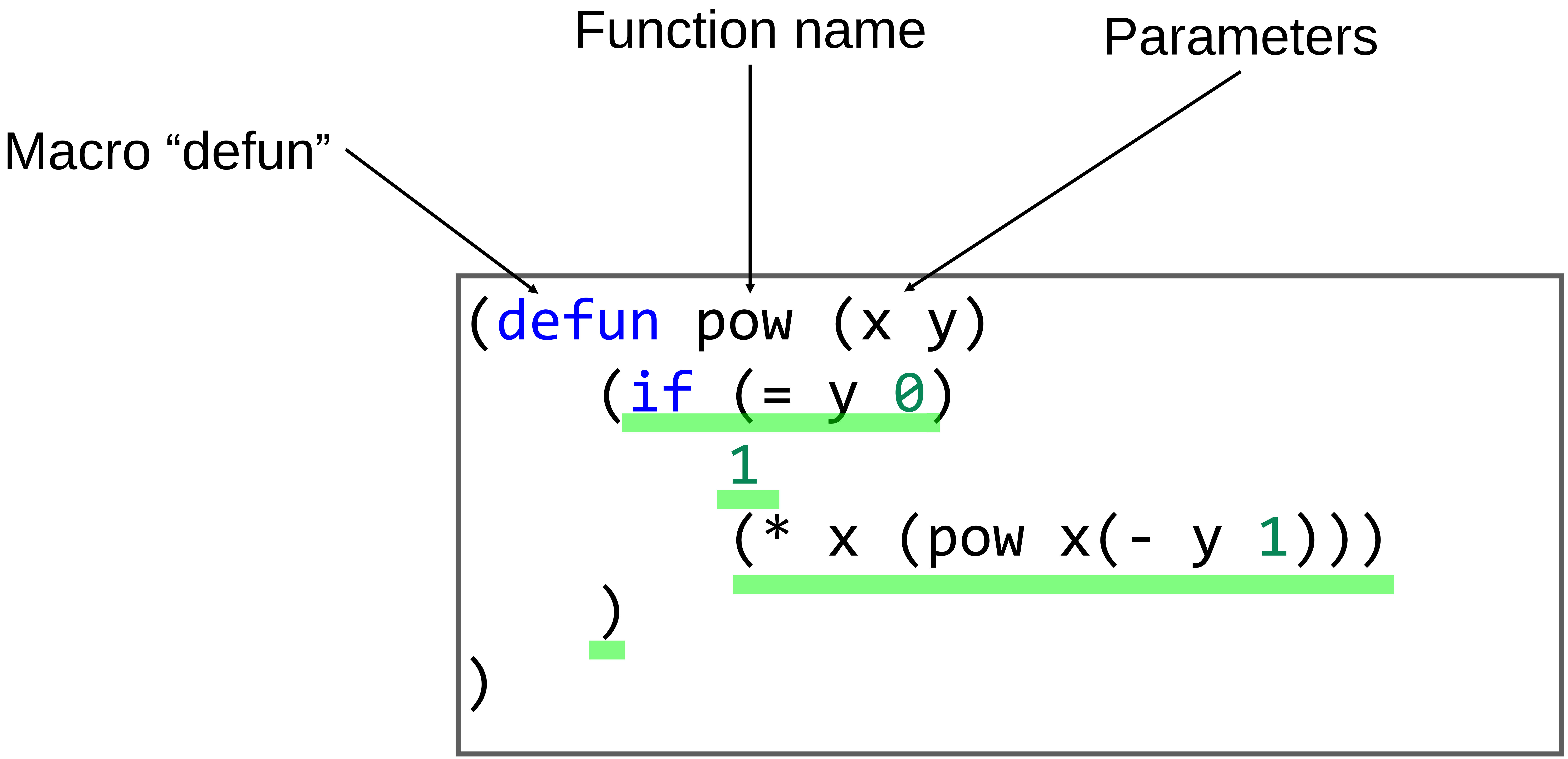
Lisp

```lisp
(defvar rad)
(defvar paint)
(defvar area)
```

Functions

Functions

Syntax



Function name

Parameters

Macro “defun”

```
(defun pow (x y)
    (if (= y 0)
        1
        (* x (pow x(- y 1)))
    )
)
```

Code Example

Circle-Paint Program

Lisp

```lisp
(defvar rad)
(defvar paint)
(defvar area)

(defun circle ()





    )
```

Code Example

Circle-Paint Program

Lisp

```
(defun circle ()
    (terpri)
    (setq rad (read))

    (setq paint (read))

    (setq area (* PI rad rad))
    )
```

Functions

Operators

Arithmetic Operations

```
(+  6 3) = 9
(- 6 3) = 3
(* 6 3) = 18
(/ 6 3) = 2
(mod 6 3) = 0
(incf 6 3) = 9
(decf 6 3) = 3
```

Comparison Operations

```
(= 3 4) = NIL
(/= 3 4) = T
(< 3 4) = T
(> 3 4) = NIL
(=< 3 4) = T
(=> 3 4) = NIL
(max 3 4) = 4
(min 3 4) = 3
```

Logical Operations

```
(and nil t) = NIL
(and 7 8) = 8
(and t t) = T
(or 7 nil) = 7
(or t nil) = T
(or nil nil) = NIL
(not nil) = T
(not T) = NIL
```

Control structures

Control structures

Decision making

- If-then-else statement
- cond: for multiple test-action clauses
- when: if-then
- case

Haskell

```
if condition then action1
             else action2
```

Lisp

```
(if (condition)
    (action1)
    (action2))
```

```
(cond (test1 action1)
    (test2 action2)
    ...
    (testN actionN))
```

```
(when (condition) (action))
```

Code Example

Circle-Paint Program

Lisp

```
(defun circle ()
    (terpri)

    (setq rad (read))

    (setq paint (read))

    (setq area (* PI rad rad)
    (if (<= area paint)
        ()
        ())
    )
```

Control structures

Loops

- loop
- loop for

```
(setq a 0)
(loop
   (setq a (+ a 1))
   (write a)
   (terpri)
   (when (> a 9) (return a))
)
```

```

(loop for a from 1 to 10
   do (print a)
)
```

```
Output:    1
           2
           3
           4
           5
           6
           7
           8
           9
           10
```

Control structures

Loops

- loop
- loop for
- do, dotimes, dolist

```
(loop for a from 1 to 10
   do (print a)
)
```

```
(loop for x in '(Toygun Sandro Tufan)
   do (format t "~s~%" x)
)
```

Output:

```
TOYGUN
SANDRO
TUFAN
```

Code Example

Circle-Paint Program

Lisp

```lisp
(defun circle ()
    (terpri)
    (write-line "Enter radius of circle: ")
    (setq rad (read))
    (format t "Enter how much paint you have: ~%")
    (setq paint (read))
    (setq area (* PI rad rad))
    (if (<= area paint)
        (format t "The circle CAN be painted!~%")
        (format t "The circle CAN NOT be painted!~%")))
```

Output:

```
Enter radius of circle:
5
Enter how much paint you have:
78
The circle CAN NOT be painted!
```

Output:

```
Enter radius of circle:
5
Enter how much paint you have:
79
The circle CAN be painted!
```

Code Example

Circle-Paint Program

Haskell

```
circle :: IO()
circle = do
  putStrLn "Enter radius of circle: "
  input1 <- getLine
  putStrLn "Enter how much paint you have: "
  input2 <- getLine
  let rad = (read input1)
  let paint = (read input2)
  let area =  (rad*rad*pi)
  if area <= paint then putStrLn "The circle CAN be painted!"
                   else putStrLn "The circle CAN NOT be painted!"
```

Haskell

```haskell
circle :: IO()
circle = do
    putStrLn "Enter radius of circle: "
    input1 <- getLine
    putStrLn "Enter how much paint you have: "
    input2 <- getLine
    let rad = (read input1)
    let paint = (read input2)
    let area =  (rad*rad*pi)
    if area <= paint then putStrLn "The circle CAN be painted!"
                     else putStrLn "The circle CAN NOT be painted!"
```

Lisp

```lisp
(defun circle ()
    (terpri)
    (write-line "Enter radius of circle: ")
    (setq rad (read))
    (format t "Enter how much paint you have: ~%")
    (setq paint (read))
    (setq area (* PI rad rad))
    (if (<= area paint)
        (format t "The circle CAN be painted with ~D m2 of paint!~%" paint )
        (format t "The circle CAN NOT be painted!~%")))
```

Conclusion

Conclusion

Lisp

- S-expressions work well with AI
- Macros and other features make it a “programmable programming language”
    - Many dialects
- Original Lisp general-purpose
- Dialects often domain-specific
    - AI, CAD, CAM, GUI etc.