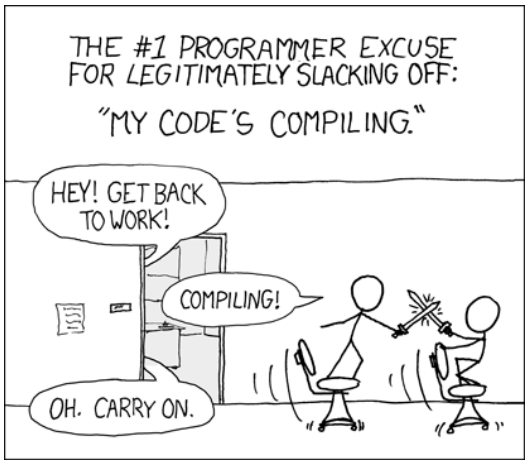
Esoteric programming languages

Useful applications,
business perspectives
and why they don´t matter

For information on proofs and theoretical background please read the documentation and ask Prof. Glaßer or employees

Reasons for programming



having fun

automating boring stuff

Making money

art?

food?

science?

love?

Chef

- Design principles
  - Program recipes should not only generate valid output, but be easy to prepare and delicious.
  - Recipes may appeal to cooks with different budgets.
  - Recipes will be metric, but may use traditional cooking measures such as cups and tablespoons.
- Programming language
  - Stack based programming
  - Turing-complete

Fibonacci Pizza.

Adding Fibonacci flavour to pizzas.

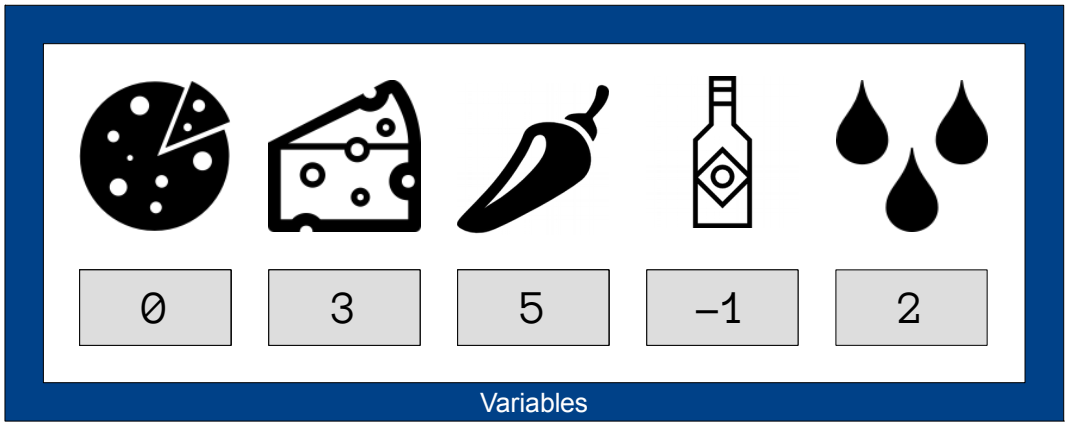
Ingredients.
pizza
1 kg cheese
1 kg Jalapenos
1 l tabasco
2 ml water

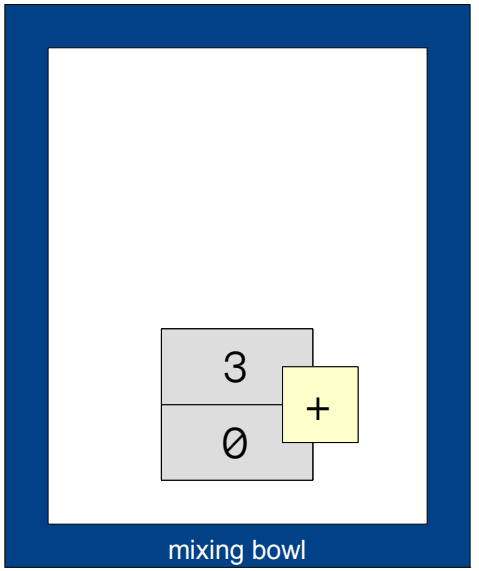
Method.
Put tabasco into mixing bowl. Remove water from mixing bowl. Fold tabasco into mixing bowl. Take pizza from refrigerator. Heat the pizza. Put cheese into mixing bowl. Add jalapenos to mixing bowl. Put jalapenos into mixing bowl. Fold cheese into mixing bowl. Fold jalapenos into mixing bowl. Put pizza into mixing bowl. Add tabasco to mixing bowl. Fold pizza into mixing bowl. Heat the pizza until hot Put pizza into mixing bowl. Add cheese to mixing bowl.

Serves 1.

Chef

```
→ 1  Fibonacci Pizza.

→ 2  Adding Fibonacci flavour to pizzas.

→ 3  Ingredients.
→ 4  pizza
→ 5  1 kg cheese
→ 6  1 kg Jalapenos
→ 7  1 l tabasco
→ 8  2 ml water

→ 9  Method.
→10  Put tabasco into mixing bowl.
→11  Remove water from mixing bowl.
→12  Fold tabasco into mixing bowl.
→13  Take pizza from refrigerator.
→14  Heat the pizza.
→15          Put cheese into mixing bowl.
→16          Add jalapenos to mixing bowl.
→17          Put jalapenos into mixing bowl
→18          Fold cheese into mixing bowl
→19          Fold jalapenos into mixing bowl
→20          Put pizza into mixing bowl.
→21          Add tabasco to mixing bowl.
→22          Fold pizza into mixing bowl.
→23  Heat the pizza until hot
→24  Put pizza into mixing bowl.
→25  Add cheese to mixing bowl.

→26  Serves 1.
```



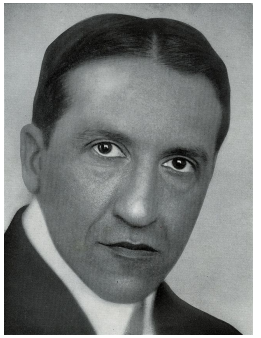


Chef

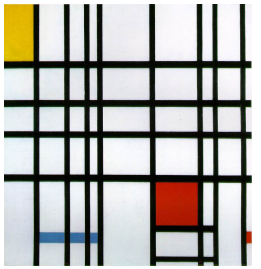
- Further Basic Instructions
  - `Put <ingredient> into 2nd mixing bowl.` `// access more stacks`
  - `Combine <ingredient> into mixing bowl.` `// multiplication`
- Writing to std:out
  - Liquid ingredients are printed as Unicode characters
    - Measures: `ml, l, dash[es]`
    - Examples: `65 = 'A', 51 = '3'`
  - Dry and unspecified ingredients are printed as numerical characters
    - Measures: `g, kg, pinch[es]`
    - Examples: `65 = '65', 3 = '3'`
- Advanced Instructions
  - `Liquefy <ingredient> .` `// dry → liquid`
  - `Pour contents of the mixing bowl into the baking dish` `// for working output`
  - `Clean mixing bowl.` `// pop entire stack`
  - `Serve with <auxiliary-recipe>` `// call subroutine`
  - `Refrigerate [for <number> hours]` `// return and print`

Piet

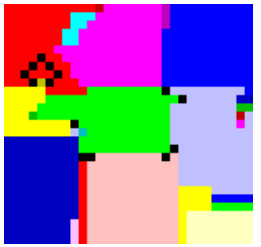
- Source code in .gif format
- Image is made out of blocks
  - Connected area → block value
  - Scaling possible: four pixel → block value = 1
- Image is traversed clockwise (with few exceptions)
  - Change in colour or hue starts an instruction
  - Black cannot be entered, white is NOOP
- Language is not Turing-complete
  - No random access to memory (or second stack)



Piet Mondrian



Composition with Red, Yellow and blue, 1921, oil painting

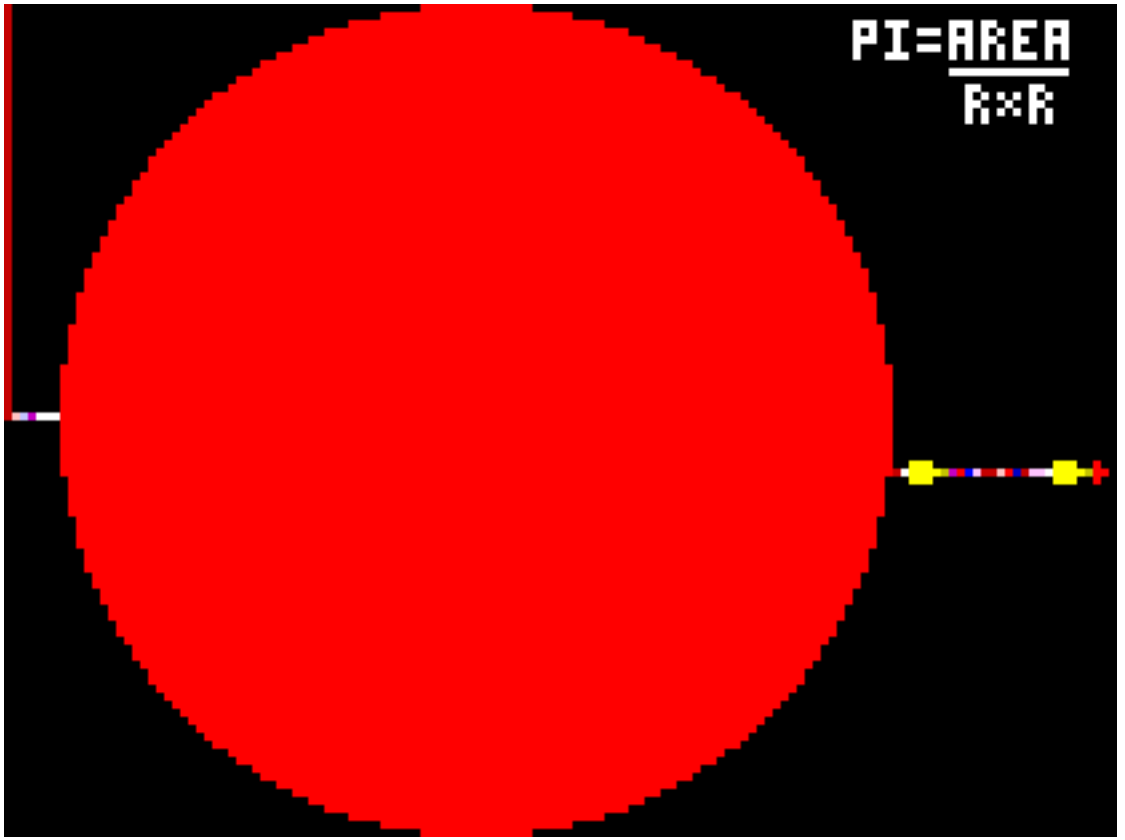


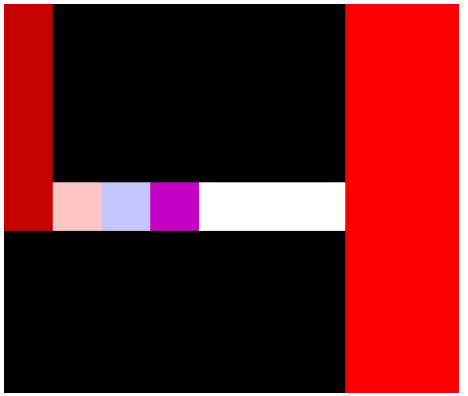
Hello World

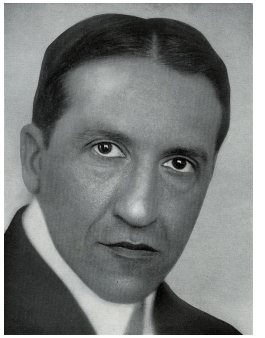
#FFC0C0 light red	#FFFFC0 light yellow	#C0FFC0 light green	#C0FFFF light cyan	#C0C0FF light blue	#FFC0FF light magenta
#FF0000 red	#FFFF00 yellow	#00FF00 green	#00FFFF cyan	#0000FF blue	#FF00FF magenta
#C00000 dark red	#C0C000 dark yellow	#00C000 dark green	#00C0C0 dark cyan	#0000C0 dark blue	#C000C0 dark magenta
#FFFFFF white			#000000 black		

	Lightness change		
Hue change	**None**	**1 Darker**	**2 Darker**
None		push	pop
1 Step	add	subtract	multiply
2 Steps	divide	mod	not
3 Steps	greater	pointer	switch
4 Steps	duplicate	roll	in(number)
5 Steps	in(char)	out(number)	out(char)

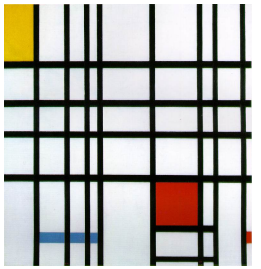
Piet



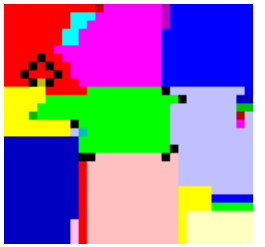




Piet Mondrian



Composition with Red, Yellow and blue, 1921, oil painting



Hello World

#FFC0C0 light red	#FFFFC0 light yellow	#C0FFC0 light green	#C0FFFF light cyan	#C0C0FF light blue	#FFC0FF light magenta
#FF0000 red	#FFFF00 yellow	#00FF00 green	#00FFFF cyan	#0000FF blue	#FF00FF magenta
#C00000 dark red	#C0C000 dark yellow	#00C000 dark green	#00C0C0 dark cyan	#0000C0 dark blue	#C000C0 dark magenta
#FFFFFF white			#000000 black		

	Lightness change		
Hue change	**None**	**1 Darker**	**2 Darker**
None		push	pop
1 Step	add	subtract	multiply
2 Steps	divide	mod	not
3 Steps	greater	pointer	switch
4 Steps	duplicate	roll	in(number)
5 Steps	in(char)	out(number)	out(char)

Shakespeare

- Language
  - The design goal was to make a language with beautiful source code that resembled Shakespeare plays. There are no fancy data or control structures, just basic arithmetic and gotos. You could say we have combined the expressiveness of BASIC with the user-friendliness of assembly language.
- Example
  - ```
    The unwanted love

    Juliet, a young woman who fell in love with Romeo
    Romeo, a young man who dislikes Juliet

    Act I: The rejection

    Scene I: The only scene with basic output

    [Enter Romeo and Juliet]

    Juliet: You are as beautiful and beloved as the sun.
    Romeo: You are as dirty as a pig.
    Juliet: Open your heart!

    [Exeunt]
    ```
  - ```
    Romeo = 2 * 2 * 1
    Juliet = 2 * -1
    numOut(Romeo)
    ```

Parodies

- HQ9+
  - “can do everything a Turing-complete language can do”
  - 99 bottles of beer on the wall, 99 bottles of beer.
    Take one down and pass it around, 98 bottles of beer on the wall.
- Whitespace
  - Language optimised for cheap printing
- Java2k
  - Probabilistic language for physicist
  - Every instruction is computed with 90% certainty
  - Numbers are base11 (approximation to base10)
  - Variables are initialized with random values
  - Memory is freed in random intervals
- Trumpscript
  - Make america great → `america = “great”`

Brainfuck

- Very simple Turing-complete language
  - MS-DOS compiler with only 98 bytes
-

index	. . .	–2	–1	0	1	2	. . .
value	. . .	0	0	0	0	0	. . .

- Instructions

`>`	`index++`	`<`	`index--`
`+`	`value++`	`-`	`value--`
`[`	`while(value){`	`]`	`}`
`,`	`value = read()`	`.`	`print(value)`

- Examples

`[<+<+>>-]`	`v[i-1] = v[i]`	`v[i-2] = v[i]`	`v[i] = 0`
`<<[->>+<<]>>`	`v[i] = v[i-2]`	`v[i-2] = 0`	
`<[>>+<<-]>`	`v[i+1] = v[i-1] + v[i+1]`		`v[i-1] = 0`

- Many derivatives

`/`	`if(cur != 0){ […] }`

Brainfuck

- Full addition example
  - ```
    >>>>>                   ptr to pos 5
    +++++ > +++ <           init with 5 and 3
    [<+<+>>-]               duplicate 5
    >[>+>+<<-]<             duplicate 3
    <<[->>+<<]>>            move 5
    >>>[-<<+>>]<<<          move 3
    <[>>>+<<<-]>            add
    ```
- Interpreter: https://fatiherikli.github.io/brainfuck-visualizer/#
- Brainfuck 2D
  - Stars indicate directions of execution
  - Sum of number indicates repetition of instruction
  - ```
    *                   **.************************
     *                  *
      **5555555555555+**
    ```

Befunge

- Simple stack-based, Turing-complete language which is hard to compile
  - Code is stored in 2D-array
  - No variables, only instructions for reading and altering the source file
- Possibility for creating animated ASCII-art in code

```
v  (635412)
 4 (offset)
 6 (length)
v                          <
 0 (curpos)
 0 (moving)
v            <              |`0-1-\g21g41<
v          p21-1g21  p41"0"  <     >$$v
>"0": 11g\-\ :12g\-\         $   :!|  @
v  \g0\ g0+1:+ -\g41"0"\+-\"0"g41  <
    >                   >14g1+:14p"0"--  $^
> `!|                   ^  p0+1\g51       <
    >"0"11g\-"0"14g\-+ : 0g15p :: 1+0g\0p ^
```

- Interpreter: http://www.quirkster.com/iano/js/befunge.html
- Further examples: https://de.wikipedia.org/wiki/Befunge

Intercal

- **C**ompiler **L**anguage **W**ith **N**o **P**ronouncable **A**cronym
- Language build to be ~~painful and difficult~~ fun and challenging
  - Turing-complete
- Slightly different versions: Intercal 72, C-Intercal, CLC-Intercal
  - Some instructions have different results depending on version
- Polite language: ¼ – ⅓ of instructions must begin with “Please”
  - **Only** possible compiler error apart from “random compile error”
- Example program: “ONE NINE THREE” → “CXCIII”

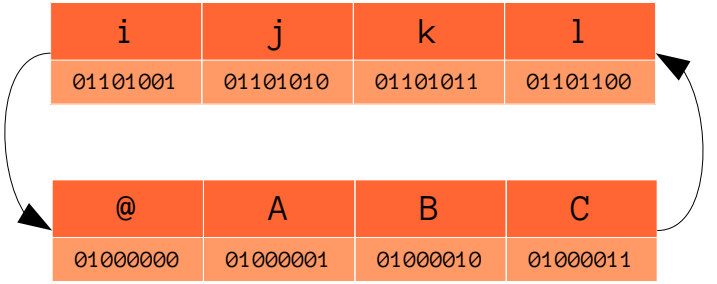
```
DO WRITE IN .1
DO READ OUT .1
PLEASE GIVE UP
```

  - Also possible in Tagalog and many other languages
- Data types and variables

`.1` 16 bit integer	`:1` 32 bit integer	I: english, O: roman
`,1` 16 bit array	`;1` 32 bit array	IO: character
`#1` the constant 1	`,1SUB#1` first 16 bit integer of first 16 bit array	

Intercal – Advanced Instructions

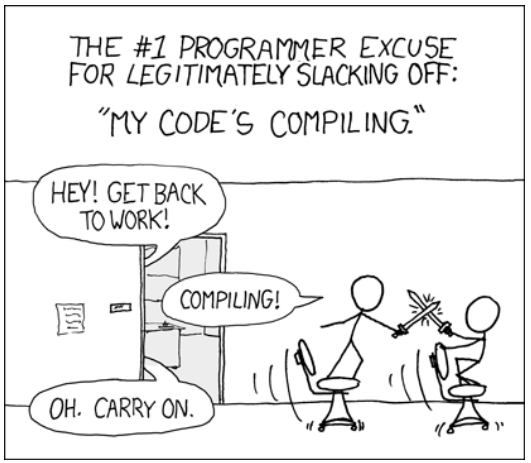
- The programmer desiring to handle input on a character basis should consider using another language.
- Basic I/O:
  - Input: `[A, C, B] → [65, 2, 255]`
  - Output: `[186, 1, 195] → [b, Q, A]`



position	0	70	69	130
rotation		(256–186)	(70–1)	(69–195+256)
ASCII		F	E	
binary	00000000	01000110	01000101	10000010
rotated	00000000	01100010	10100010	01000001
dec	0	98	81	65
ASCII		b	Q	A

- Logical Operations: $26_{10} = 11010_2$, $\#\&26 = 16_{10} = 01000_2$
- Control or data structures are not part of language
  - `NEXT` stack instead of instruction pointer
  - Manipulation of instruction stack as flow control: `RESUME`

Reasons for programming



having fun

automating boring stuff

Making money

art?

food?

science?

love?

Conclusion

- Source: Internet
- Licence informations
  - Chili & Tabasco Icon made by Freepick from www.flaticon.com
  - Water Icon made by Daniel Bruce (216782, Iconfinder)
  - Pizza Icon made by Danilo Demarco (171473, Iconfinder)
  - Cheese Icon made by Zlatko Najdenovski (809009, Iconfinder)
- Read more
  - https://de.wikipedia.org/wiki/Liste_von_Hallo-Welt-Programmen/Sonstige#Esoterische_Programmiersprachen
  - http://divingintointercal.blogspot.de/2007/03/good-programmer-constantly-strives-to.html
- Get involved
  - https://esolangs.org/wiki/Main_Page
  - ANTLR