TDP007 - Konstruktion av datorspråk

Utblickar

Pontus Haglund

Institutionen för datavetenskap

LINKÖPING UNIVERSITY

1. Stack-maskin
2. Programspråk ni redan känner igen
3. Lite nya programspråk!
4. Lågnivåspråk
5. Högnivåspråk
6. Verktyg för att jobba med nya språk

Dagens föreläsning

- Introduktion till stack-maskiner
  - Övning: Implementera delar av en stack-maskin
- Överblick av olika programspråk och deras användningsområde och syften
  - Övning: Utöka befintlig kod i helt nya programspråk!
- Lite tips på verktyg som kan vara bra i framtiden!

1 Stack-maskin
2 Programspråk ni redan känner igen
3 Lite nya programspråk!
4 Lågnivåspråk
5 Högnivåspråk
6 Verktyg för att jobba med nya språk

LINKÖPING UNIVERSITY

Stack-maskin

Vad är en stackmaskin?

- En teoretisk beräkningsmodell där minnet i princip endast består av en stack
- Ett program som simulerar en sådan maskin

Stack-maskin

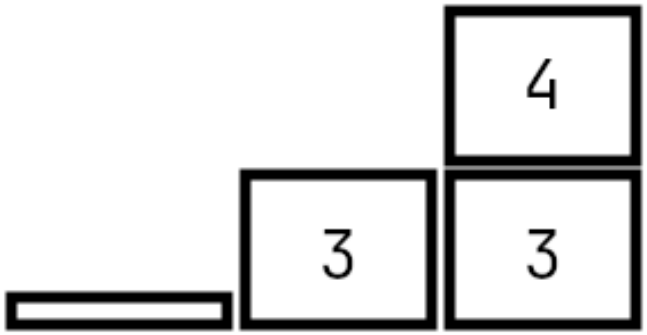
```
Tokens:  3 4 5 + * PRINT
```

Stack-maskin

```
Tokens:  3 4 5 + * PRINT
```

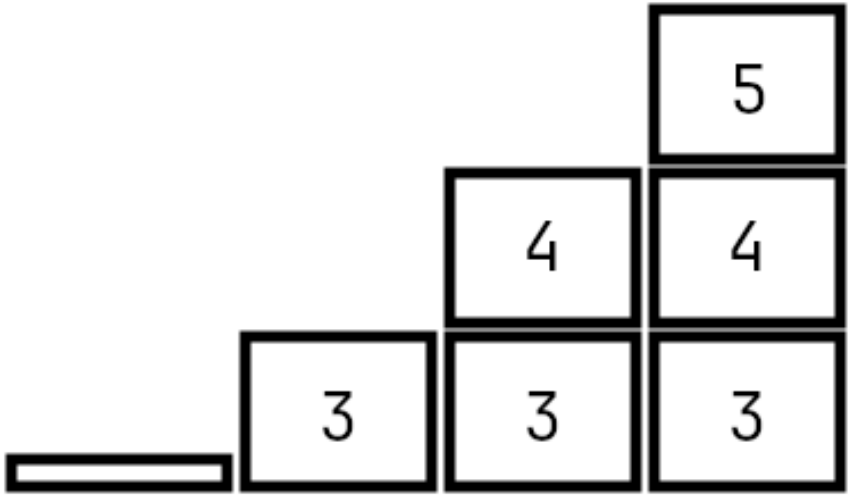
Stack-maskin

```
Tokens:  3 4 5 + * PRINT
```

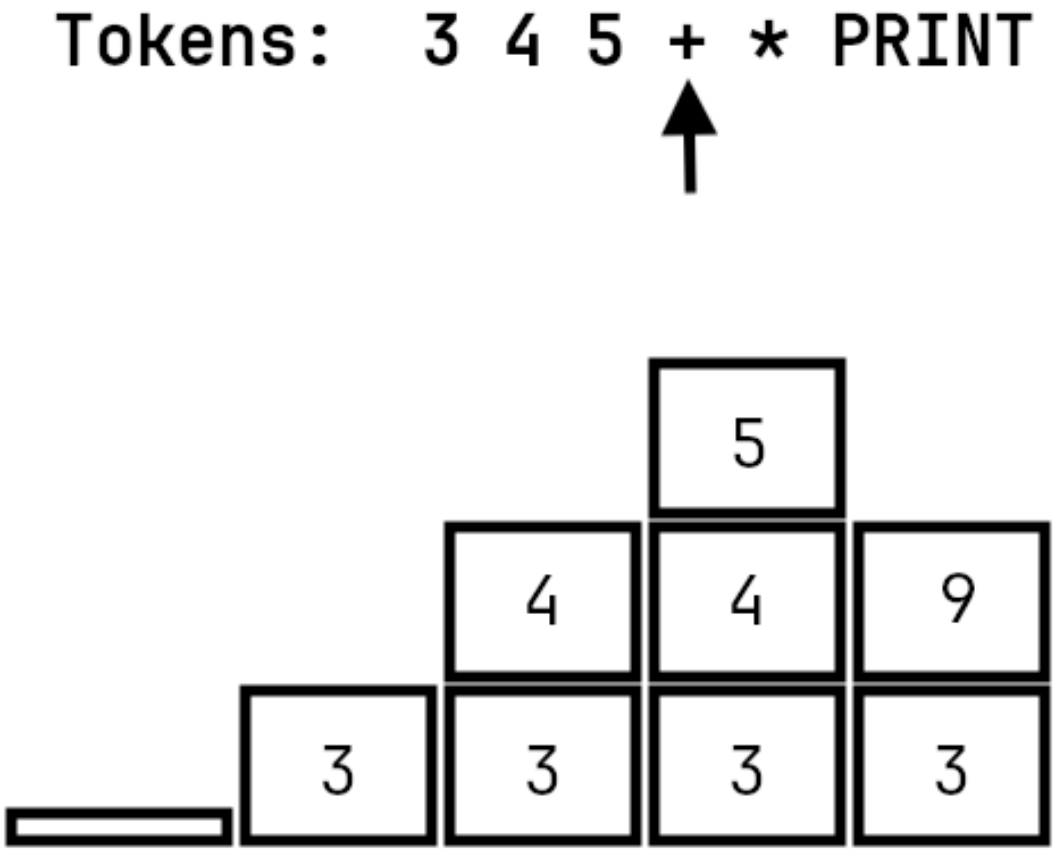


Stack-maskin

```
Tokens:  3 4 5 + * PRINT
```

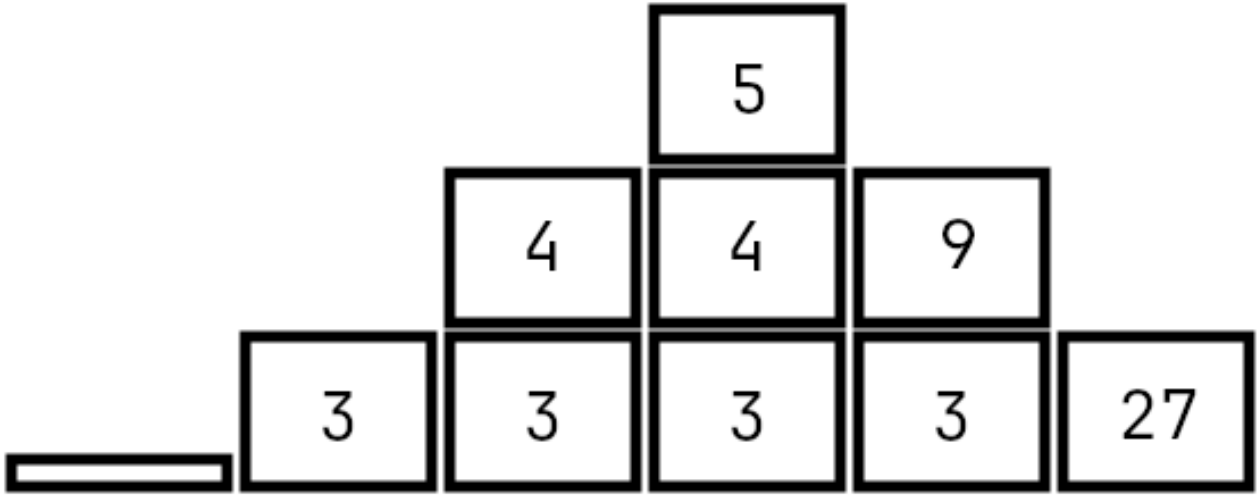


Stack-maskin



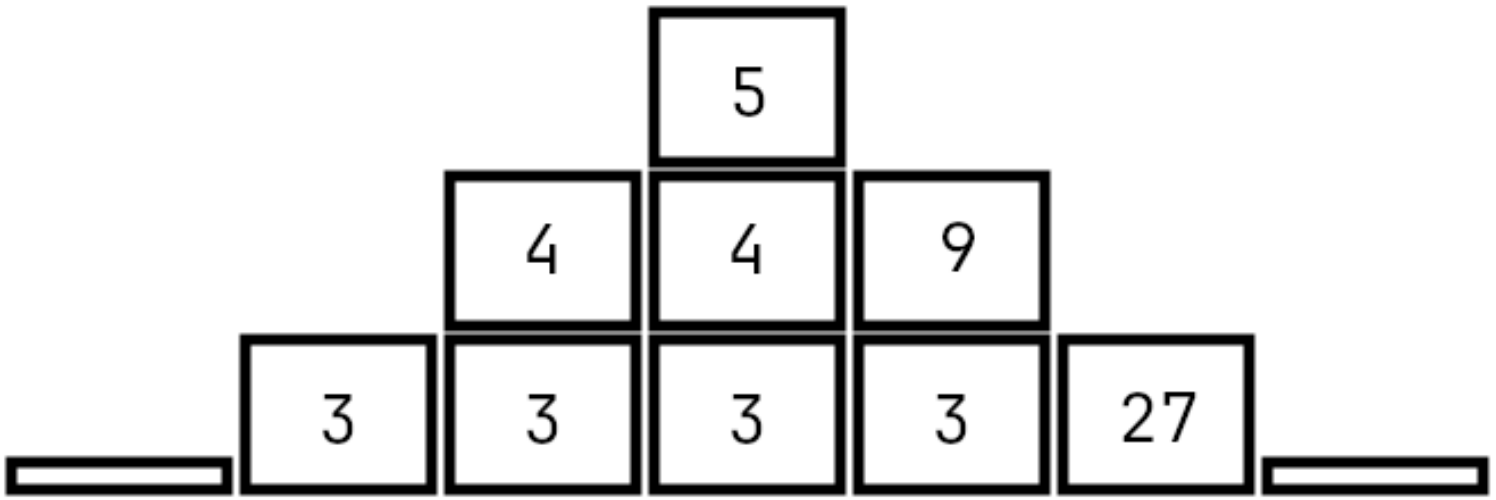
Stack-maskin

```
Tokens:  3 4 5 + * PRINT
```



Stack-maskin

```
Tokens:  3 4 5 + * PRINT
Output: 27
```



Övning

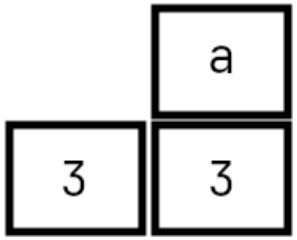
- Vi kommer att arbeta med den givna filen `stack.rb`
- Denna implementation ska vi utöka med variabler!
- Vi tittar på hur detta kan se ut...

Stack-maskin

```
Tokens:  3 a SET 4 a GET 5 + * PRINT
```

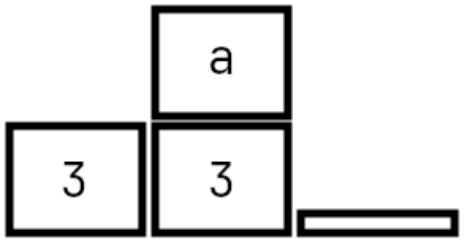
Stack-maskin

```
Tokens:  3 a SET 4 a GET 5 + * PRINT
```



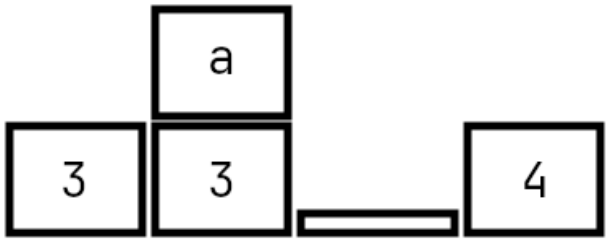
Stack-maskin

```
Tokens:  3 a SET 4 a GET 5 + * PRINT   Variabler:
                                       { a: 3 }
```



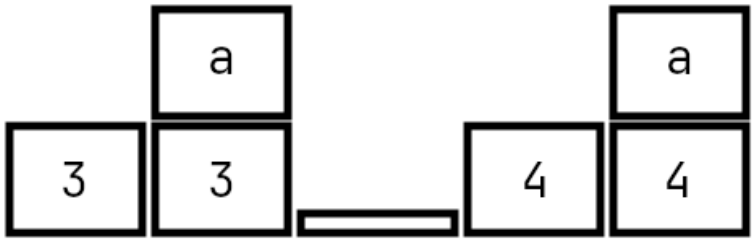
Stack-maskin

```
Tokens:  3 a SET 4 a GET 5 + * PRINT   Variabler:
                                       { a: 3 }
```



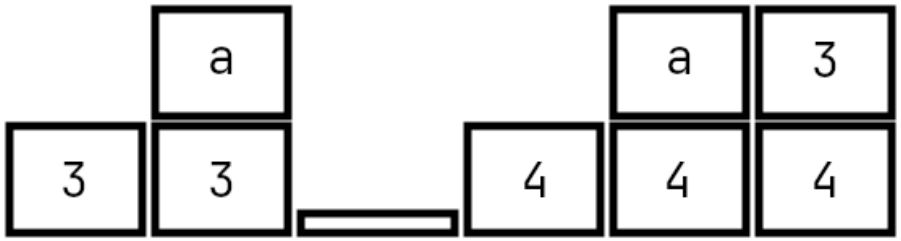
Stack-maskin

```
Tokens:  3 a SET 4 a GET 5 + * PRINT   Variabler:
                                       { a: 3 }
```



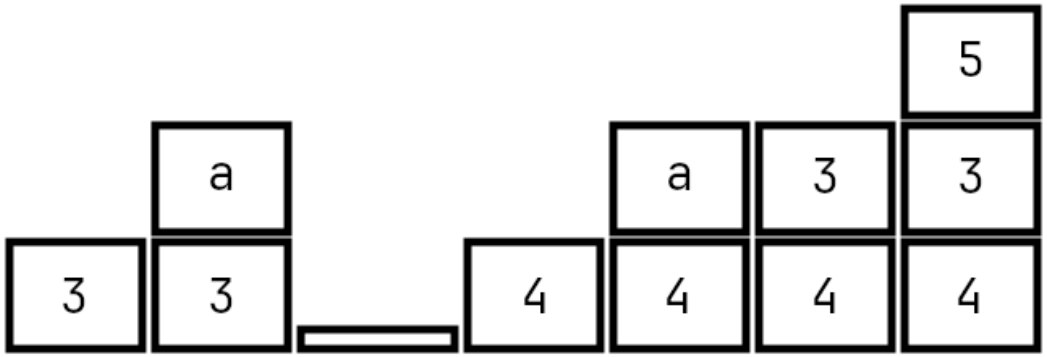
Stack-maskin

```
Tokens:  3 a SET 4 a GET 5 + * PRINT   Variabler:
                                       { a: 3 }
```



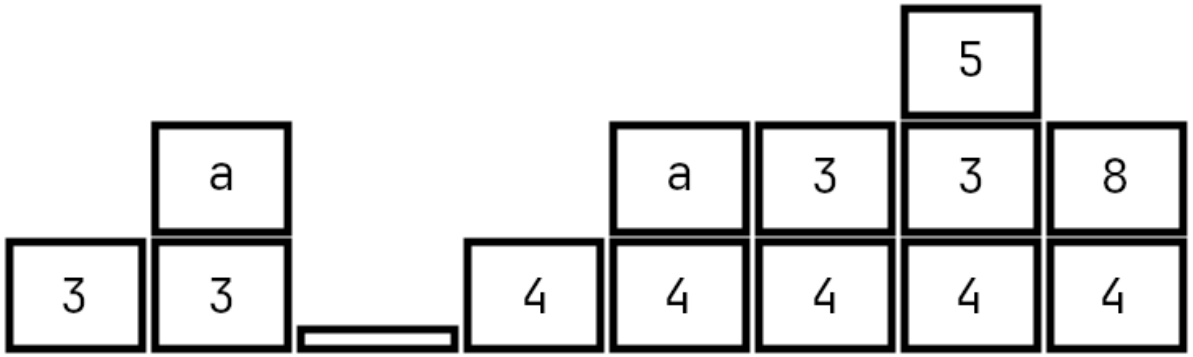
Stack-maskin

```
Tokens:  3 a SET 4 a GET 5 + * PRINT   Variabler:
                                       { a: 3 }
```



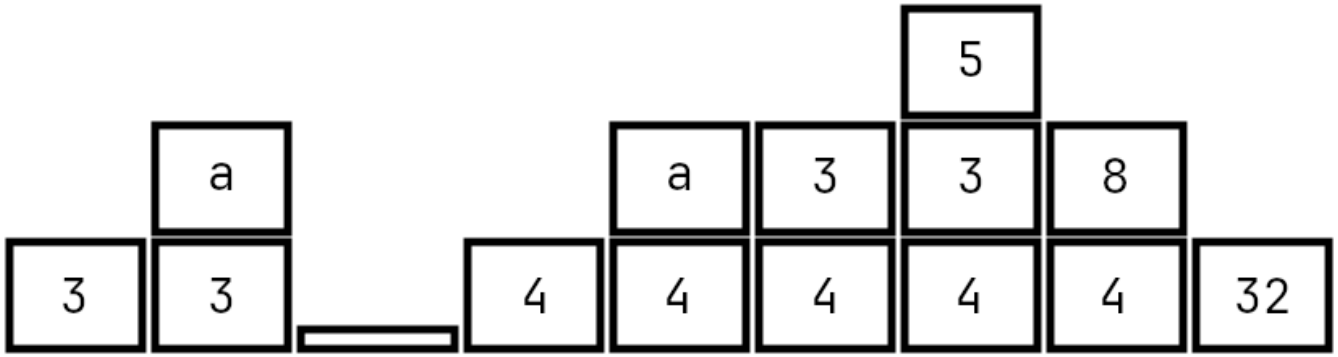
Stack-maskin

```
Tokens:  3 a SET 4 a GET 5 + * PRINT   Variabler:
                                       { a: 3 }
```



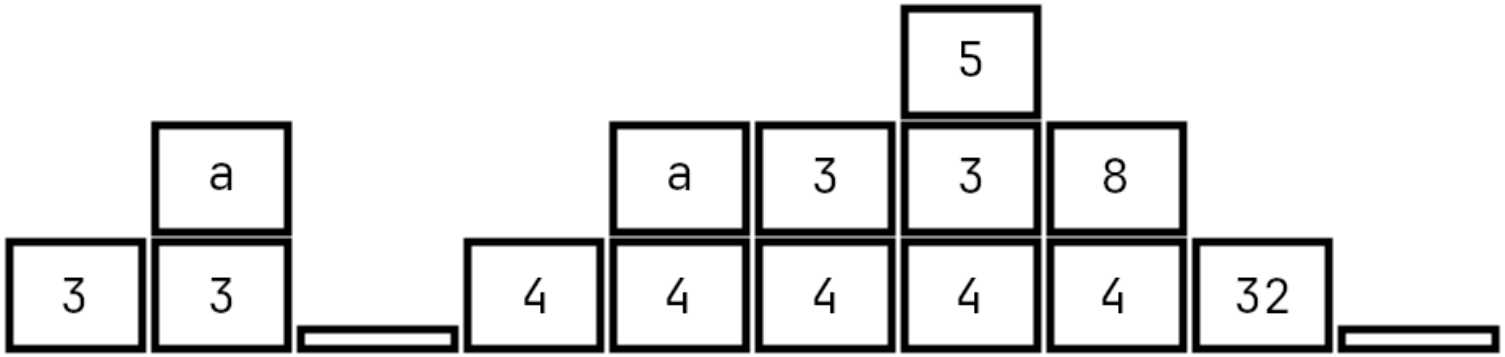
Stack-maskin

```
Tokens:  3 a SET 4 a GET 5 + * PRINT   Variabler:
                                       { a: 3 }
```



Stack-maskin

```
Tokens:  3 a SET 4 a GET 5 + * PRINT   Variabler:
Output: 32                             { a: 3 }
```



Övning

- Bygg ut stackmaskinen i `stack.rb` med:
- `SET`:
  - Poppar två element från stacken
  - Första elementet är variabelnamnet, andra är värdet
- `GET`:
  - Poppar ett element från stacken
  - Slår upp elemntet i variabellistan och pushar det till stacken

1 Stack-maskin
2 Programspråk ni redan känner igen
3 Lite nya programspråk!
4 Lågnivåspråk
5 Högnivåspråk
6 Verktyg för att jobba med nya språk

LINKÖPING UNIVERSITY

Ruby

```
def manipulate(arr)
  new_arr = []
  arr.each do |x|
    new_arr << x + 5
  end
  new_arr
end

puts "Hello world!"

my_list = [10, 15, 20, 25, 30]
result = manipulate(my_list)
puts "Result: #{result}"
```

C++

```
#include <iostream>
#include <vector>

using namespace std;

vector<int>
  manipulate(const vector<int>& arr) {
  vector<int> newArr;
  for (int x : arr) {
    newArr.push_back(x + 5);
  }
  return newArr;
}
```

C++

```
int main() {
  cout << "Hello world!" << endl;
  vector<int> myList = {10, 15, 20, 25, 30};
  vector<int> result = manipulate(myList);

  cout << "Result: ";
  for (int x : result) {
    cout << x << " ";
  }
  cout << std::endl;

  return 0;
}
```

Python

```python
def manipulate(arr):
    new_arr = []
    for x in arr:
        new_arr.append(x + 5)
    return new_arr


if __name__ == '__main__':
    print("Hello world!")
    my_list = [10, 15, 20, 25, 30]
    result = manipulate(my_list)
    print("Result:", result)
```

1 Stack-maskin
2 Programspråk ni redan känner igen
3 Lite nya programspråk!
4 Lågnivåspråk
5 Högnivåspråk
6 Verktyg för att jobba med nya språk

LINKÖPING UNIVERSITY

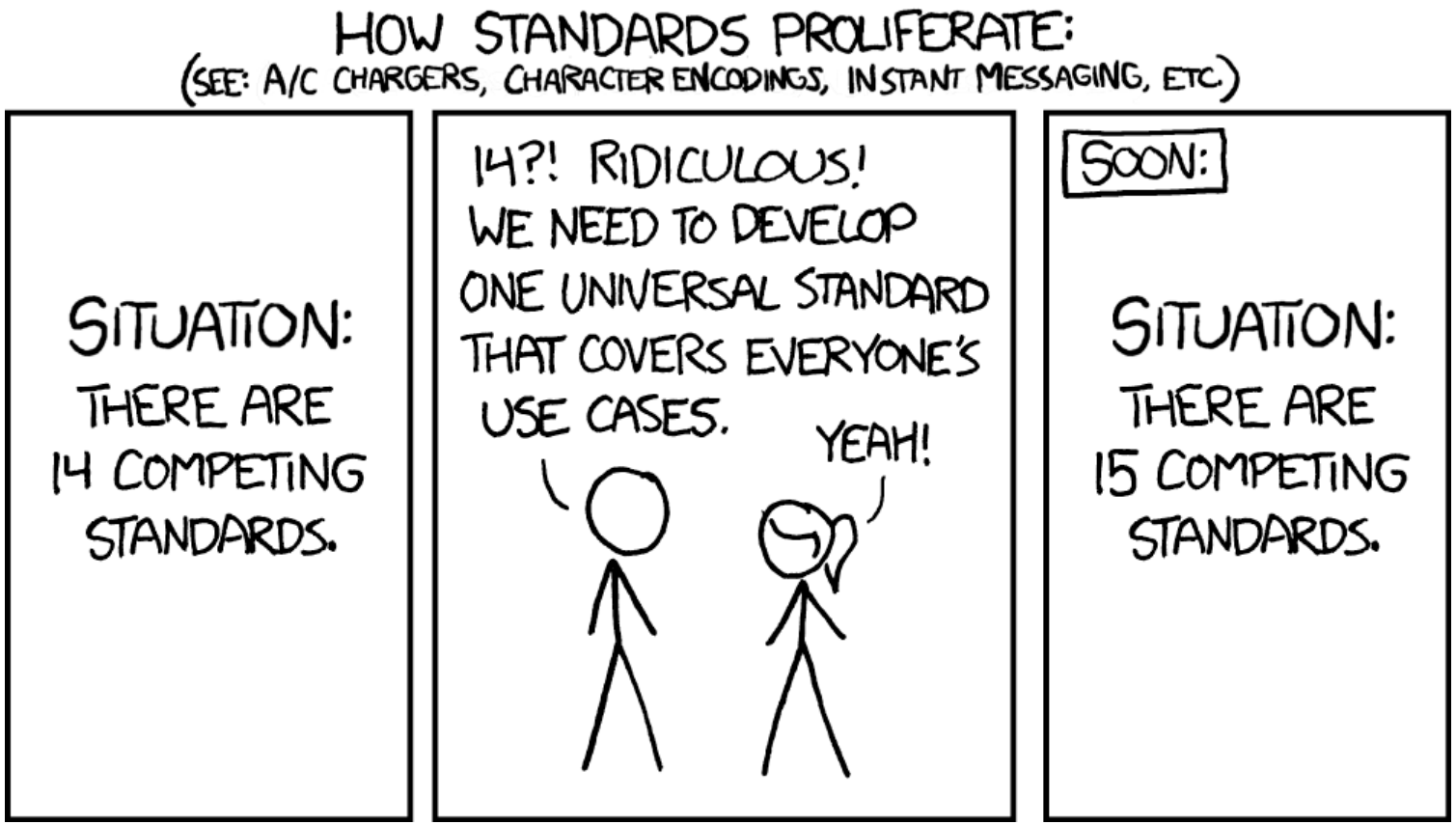
Varför skapas nya programspråk?

- Nya problem att lösa
- Gamla problem man vill slippa
- Nya tekniker som kan användas
- Gamla tekniker som är bökiga att använda
- Bara för att man kan

Bython

```
def manipulate(arr) end
  new_arr = []
  for x in arr end
    new_arr.append(x + 5)
  end
  return new_arr
end

if __name__ == '__main__' end
  print("Hello world!")
  my_list = [10, 15, 20, 25, 30]
  result = manipulate(my_list)
  print("Result:", result)
end
```


HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC.)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.

Lågnivå vs. högnivå

- Språk kan klassas som högnivå eller lågnivå
- Inte strikta definitioner, väldigt subjektivt
- Vi tar upp några generella saker som skiljer dem, men detta är ingen exakt definition

1 Stack-maskin
2 Programspråk ni redan känner igen
3 Lite nya programspråk!
4 Lågnivåspråk
5 Högnivåspråk
6 Verktyg för att jobba med nya språk

LINKÖPING UNIVERSITY

Lågnivåspråk

- Jobbar nära hårdvaran
- Snabb exekvering
- Mer fokus på minneshantering
- Har (generellt) färre abstraktioner
- Kan separereas vidare till lågnivå / systemprogramspråk
- Kompileras generellt ned till en binärfil

C

```
#include <stdio.h>

void manipulate
  (const int arr[], int size, int out[]) {
  for (int i = 0; i < size; i++) {
    out[i] = arr[i] + 5;
  }
}
```

C

```c
int main() {
  printf("Hello world!\n");
  int myList[5] = {10, 15, 20, 25, 30};
  int result[5];

  manipulate(myList, 5, result);

  printf("Result: ");
  for (int i = 0; i < 5; i++) {
    printf("%d ", result[i]);
  }
  printf("\n");
  return 0;
}
```

Rust

```
fn manipulate(arr: &Vec<i32>) -> Vec<i32> {
  let mut new_arr = Vec::new();
  for &x in arr.iter() {
    new_arr.push(x + 5);
  }
  new_arr
}

fn main() {
  println!("Hello world!");
  let my_list = vec![10, 15, 20, 25, 30];
  let result = manipulate(&my_list);
  println!("Result: {:?}", result);
}
```

Go (Golang)

```go
package main
import "fmt"

func manipulate(arr []int) []int {
  newArr := make([]int, len(arr))
  for i, v := range arr {
    newArr[i] = v + 5
  }
  return newArr
}
```

Go (Golang)

```
func main() {
  fmt.Println("Hello world!")
  myList := []int{10, 15, 20, 25, 30}
  result := manipulate(myList)
  fmt.Println("Result:", result)
}
```

1 Stack-maskin
2 Programspråk ni redan känner igen
3 Lite nya programspråk!
4 Lågnivåspråk
5 Högnivåspråk
6 Verktyg för att jobba med nya språk

LINKÖPING UNIVERSITY

Högnivåspråk

- ”Gömmer” vissa detaljer om hårdvaran bakom abstraktioner
- Har oftast inte manuell minneshantering, använder garbage collectors
- Erbjuder ofta trevliga abstraktioner för vanliga saker (t.ex. nätverksanrop, kommunikation med operativsystemet)
- Bygger ofta på språk på lägre nivå (t.ex. Ruby bygger på C)

Lisp

```
(defun manipulate (lst)
  (let ((result '()))
    (dolist (x lst (reverse result))
      (push (+ x 5) result))))

(defun main ()
  (format t "Hello world!~%")
  (let* ((my-list '(10 15 20 25 30))
         (result (manipulate my-list)))
    (format t "Result: ~a~%" result)))

(main)
```

C#

```
using System;

class Program {
  static int[] Manipulate(int[] arr) {
    int[] newArr = new int[arr.Length];
    for (int i = 0; i < arr.Length; i++) {
      newArr[i] = arr[i] + 5;
    }
    return newArr;
  }
```

C#

```
        static void Main() {
          Console.WriteLine("Hello world!");
          int[] myList = {10, 15, 20, 25, 30};
          int[] result = Manipulate(myList);
          Console.Write("Result: ");
          foreach (int x in result) {
            Console.Write(x + " ");
          }
          Console.WriteLine();
        }
      }
```

Java

```
public class Main {
  public static int[]
    manipulate(int[] arr) {
    int[] newArr = new int[arr.length];
    for (int i = 0; i < arr.length; i++) {
      newArr[i] = arr[i] + 5;
    }
    return newArr;
  }
```

Java

```
    public static void main(String[] args) {
      System.out.println("Hello world!");
      int[] myList = {10, 15, 20, 25, 30};
      int[] result = manipulate(myList);
      System.out.print("Result: ");
      for (int x : result) {
        System.out.print(x + " ");
      }
      System.out.println();
    }
  }
```

JavaScript

```
function manipulate(arr) {
  const newArr = [];
  for (let i = 0; i < arr.length; i++) {
    newArr.push(arr[i] + 5);
  }
  return newArr;
}

console.log("Hello world!");

const myList = [10, 15, 20, 25, 30]
const result = manipulate(myList);
console.log("Result:", result)
```

TypeScript

```
function
  manipulate(arr: number[]): number[] {
    const newArr: number[] = []
    for (let i = 0; i < arr.length; i++) {
      newArr.push(arr[i] + 5)
    }
    return newArr;
}

console.log("Hello world!")

const myList: number[] = [10, 15, 20, 25, 30]
const result: number[] = manipulate(myList);
console.log("Result:", result);
```

1 Stack-maskin
2 Programspråk ni redan känner igen
3 Lite nya programspråk!
4 Lågnivåspråk
5 Högnivåspråk
6 Verktyg för att jobba med nya språk

LINKÖPING UNIVERSITY

Verktyg för att jobba med nya språk

- Dokumentation är A och O för att lära sig nya språk!
- Finns även Language Server Protocols (LSP) till de flesta språk som kan ge hints om vad som händer i koden medans man skriver, och om någonting gått fel
  - Körs medans man jobbar i filen, ger lite live-hints och debugging
  - Finns för i princip alla editors, t.ex. Emacs, Vim / NeoVim, VSCode m.fl.
- Formatters kan hjälpa till att hålla koden snygg, och att följa ett språks kodstandard

www.liu.se

LINKÖPING
UNIVERSITY