

# Scalar Data Numbers

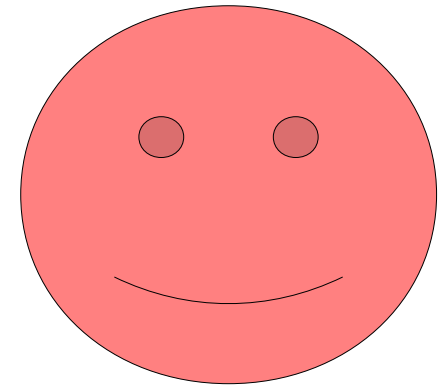
Podstawy programowania

Jolanta Bachan  
2007-10-17

# From last week...

# My first Perl program

Hello world!



# My second Perl program

Hello, Students!

# My third Perl program

Hello, Students! ...  
*(Are you sure I made you special?)*

# Basic Perl syntax

- All lines of code in Perl must end with one of two things: a semicolon or curly braces:
  - `print "Hello.";`
  - `sub {print "Hello."}`
- A line of code, called a **statement**, is basically a single instruction to the computer; it says, "Do x." It can extend over more than one line of the page.
- Anything in `{ }` is called a **block**. A block can contain several statements.
- Any line that begins with the character `#` is treated by Perl as a comment and is ignored.

# Scalar data

## Numbers

# What is scalar data?

Scalar data is the simplest kind of data. It can be either a number (like 2007 or 3.25e20) or a string of characters (like “Hello!” or an address).

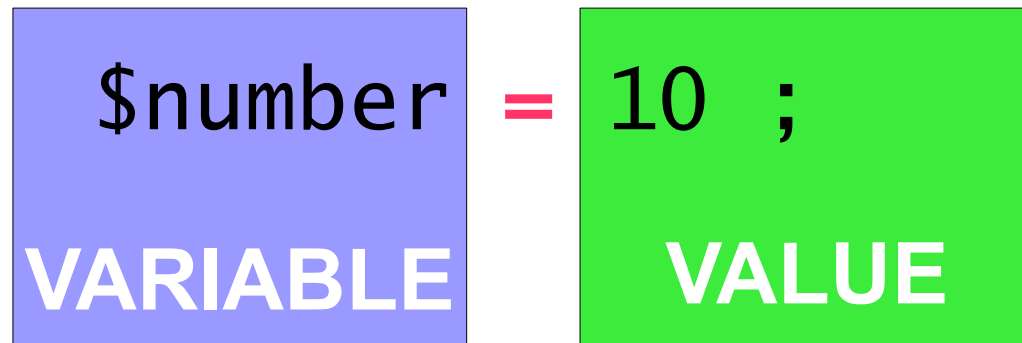
Scalar variables always begin with the “\$” character.

```
$number = 10 ;
```

# What is scalar data?

Scalar data is the simplest kind of data. It can be either a number (like 2007 or 3.25e20) or a string of characters (like “Hello!” or an address).

Scalar variables always begin with the “\$” character.



# Naming variables

- As long as you wish
- A combination of letters and underscores, but numbers are also allowed
- “\$” cannot be followed by a number
- Choose meaningful names:  
`$sum_of_two_numbers ;`

```
$x = 1 ;  
$y = 2 ;  
$z = $x + $y ;
```

**BAD!!!**

```
$x = 1 ;  
$y = 2 ;  
$sum = $x + $y ;
```

**better**

```
$first_number = 1 ;  
$second_number = 2 ;  
$sum = $first_number  
+ $second_number ;
```

**IDEAL!!!**

# Numbers

- A *literal* is the way a value is represented in the text of Perl program. You could also call this *constant* in your program.
- Literał określa sposób, w jaki wartość reprezentowana jest w kodzie źródłowym programu.
- Float literals – literały zmiennoprzecinkowe
- Integer literals – literały całkowitoliczbowe

# Float literals

- 1.25
- 7.25e45 oznacza 7.25 razy 10 do potęgi 45
- -6.5e24 oznacza -6,5 razy 10 do potęgi 24 (duża liczba ujemna)
- -12e-24 oznacza -12 razy 10 do potęgi -24 (bardzo mała liczba ujemna)
- -1.2E-23 to inny sposób zapisu liczby powyżej – litera E może być wielka

# Integer literals

- 12
- 15
- -2007
- 189099

# Integer literals

- Don't start the numbers with 0, because Perl supports octal and hexadecimal literals
  - 0377 oznacza liczbę 377 w systemie ósemkowym, równoważną liczbie 255 w systemie dziesiętnym
  - 0xff oznacza liczbę FF w systemie szesnastkowym, równoważną liczbie 255 w systemie dziesiętnym
  - 0b11111111 oznacza liczbę w systemie dwójkowy - Również liczba 255 w systemie dziesiętnym

# Scalar operators

- Operators for numbers
  - +, -, \*, /, \*\*, sqrt
- Numeric comparison operators
  - <, >, <=, >=, ==, !=

35 != 30 + 5 to Fałsz

35 == 35.0 to Prawda

# Examples

- $2 + 3$  oznacza 2 plus 3 lub inaczej 5
- $5.1 - 2.4$  oznacza 5,1 minus 2,4 lub inaczej 2,7
- $3 * 12$  oznacza 3 razy 12 równa się 36
- $14 / 2$  oznacza 14 dzielone przez 2 lub inaczej 7
- $10.2 / 0.3$  oznacza 10,2 dzielone przez 0,3 lub inaczej 34
- $10 / 3$  - Zawsze dzielenie zmiennoprzecinkowe, więc daje w rezultacie 3,3333333...

# Example

```
$a = 17;
```

```
$b = $a + 3 ; # 20
```

```
$b = $b * 2 ; # 40
```

```
$b = $b / 10 ; # 4
```

```
$b = $b**2 ; # 16
```

```
$b = sqrt($b) ; # 4
```

# Binary assignment operators

(Dwuargumentowe operatory przypisania)

- `$fred = 5 ;` ; “=” to operator przypisania
- `$fred = $fred + 5;` to zapis bez dwuargumentowego operatora przypisania
- `$fred += 5;` to zapis z dwuargumentowym operatorem przypisania
- `$barney = $barney * 3;`
- `$barney *= 3;`

# Binary Assignment Operators

- ++ -- autoinkrementacja, autodekrementacja)

`$a++ ;`

`$b-- ;`

# Logical Operators

- `&&` (logical and)
- `||` (logical or)

# Standard Input

- `<STDIN>` - Standard input as scalar data

# Standard Output

- `print` function

# Use strict ;

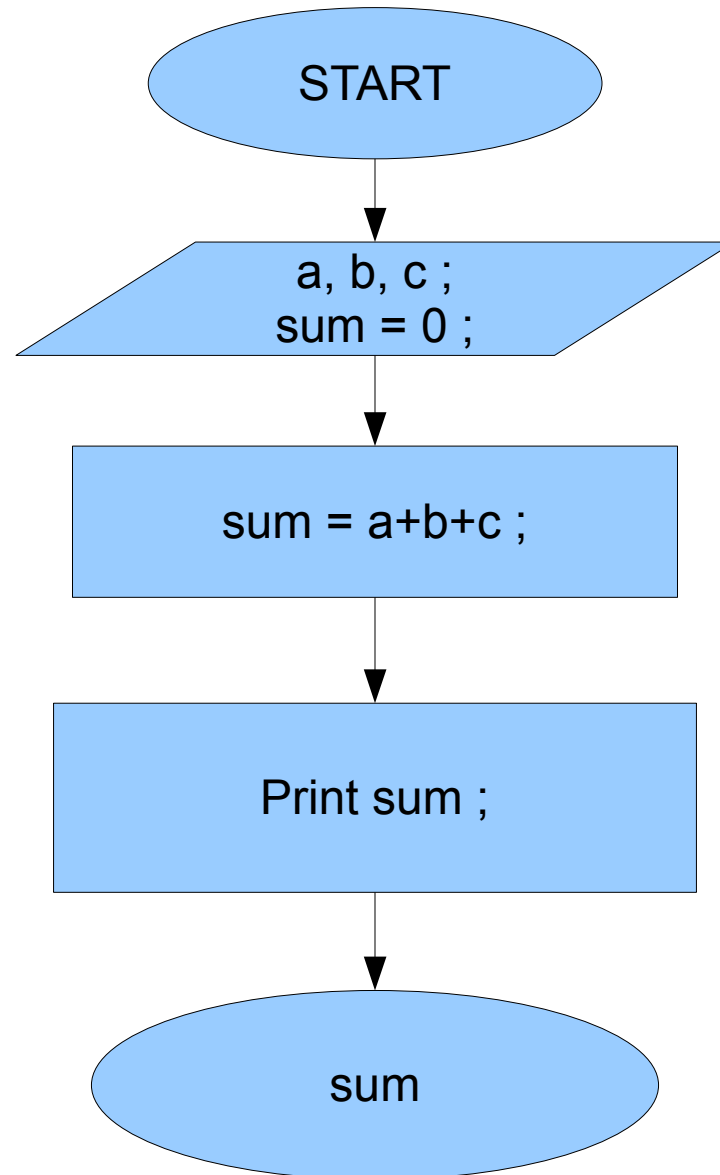
- the program will run slightly faster (variables created with my are accessed slightly faster than ordinary variables)
- You will catch mistakes in typing faster

```
use strict;  
my $a = 5 ;  
my $b ;  
$b = $a * 2 ;
```

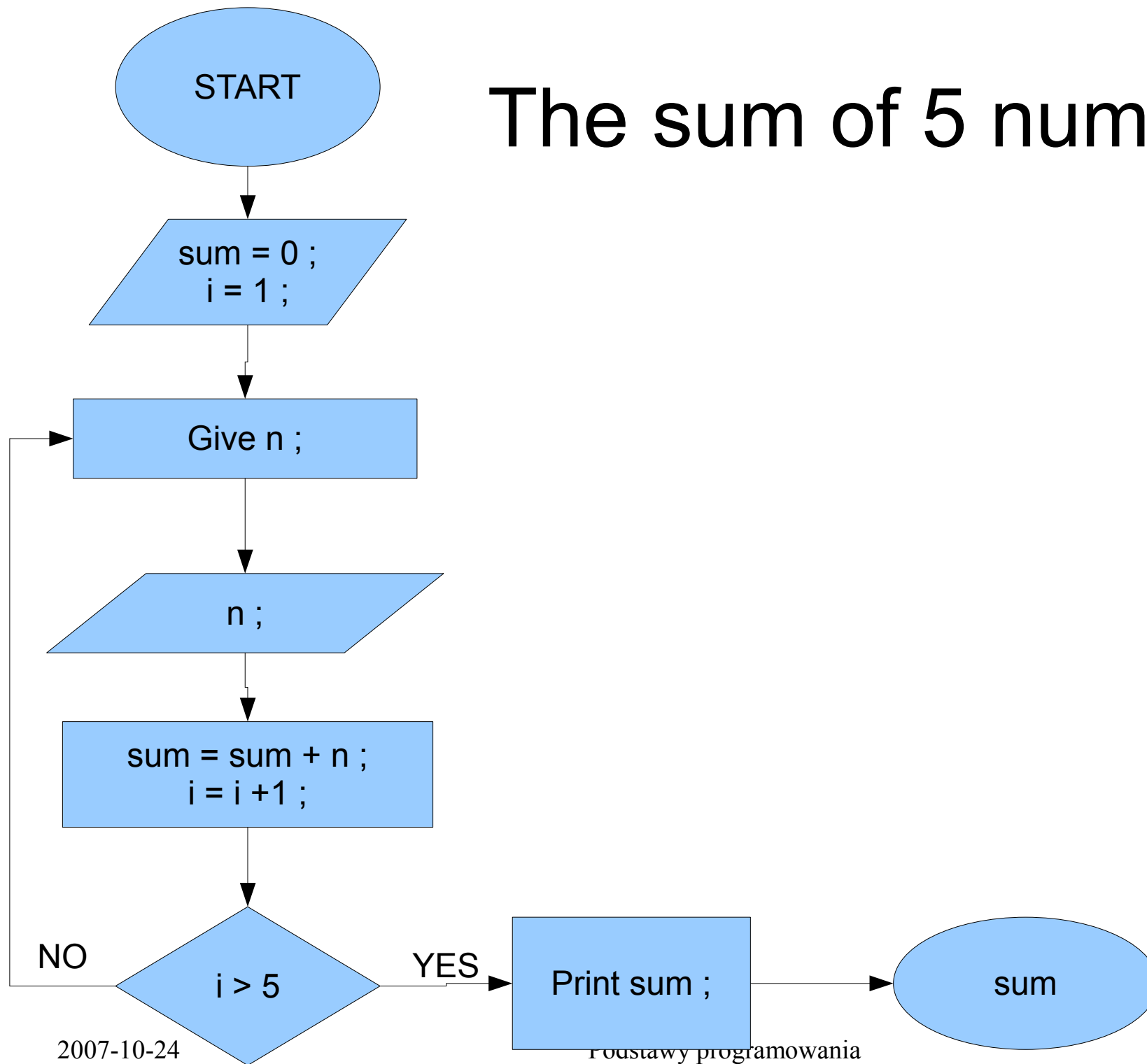
# Exercise

- Implement the algorithms we have designed so far in Perl.

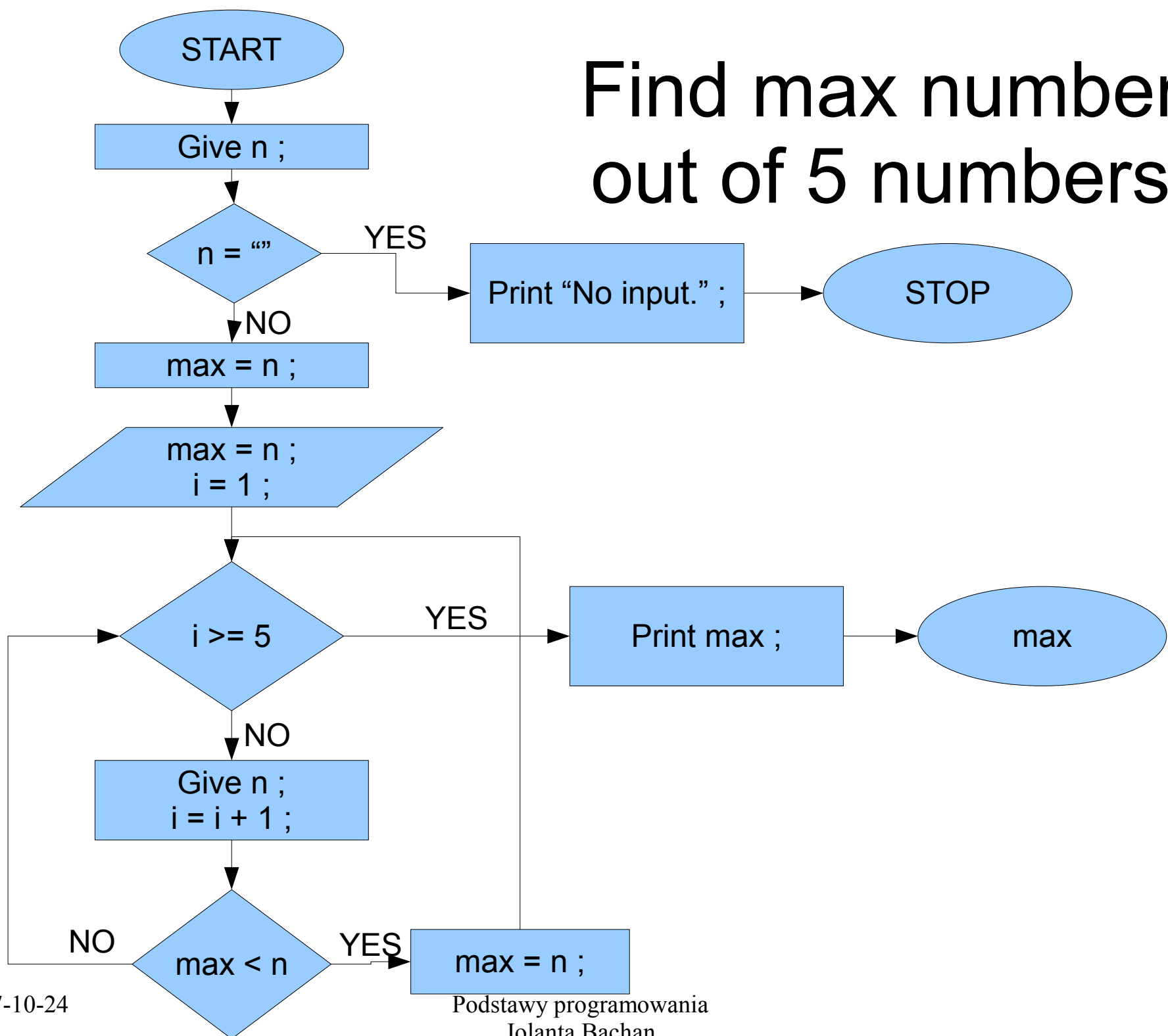
# The sum of a, b, c



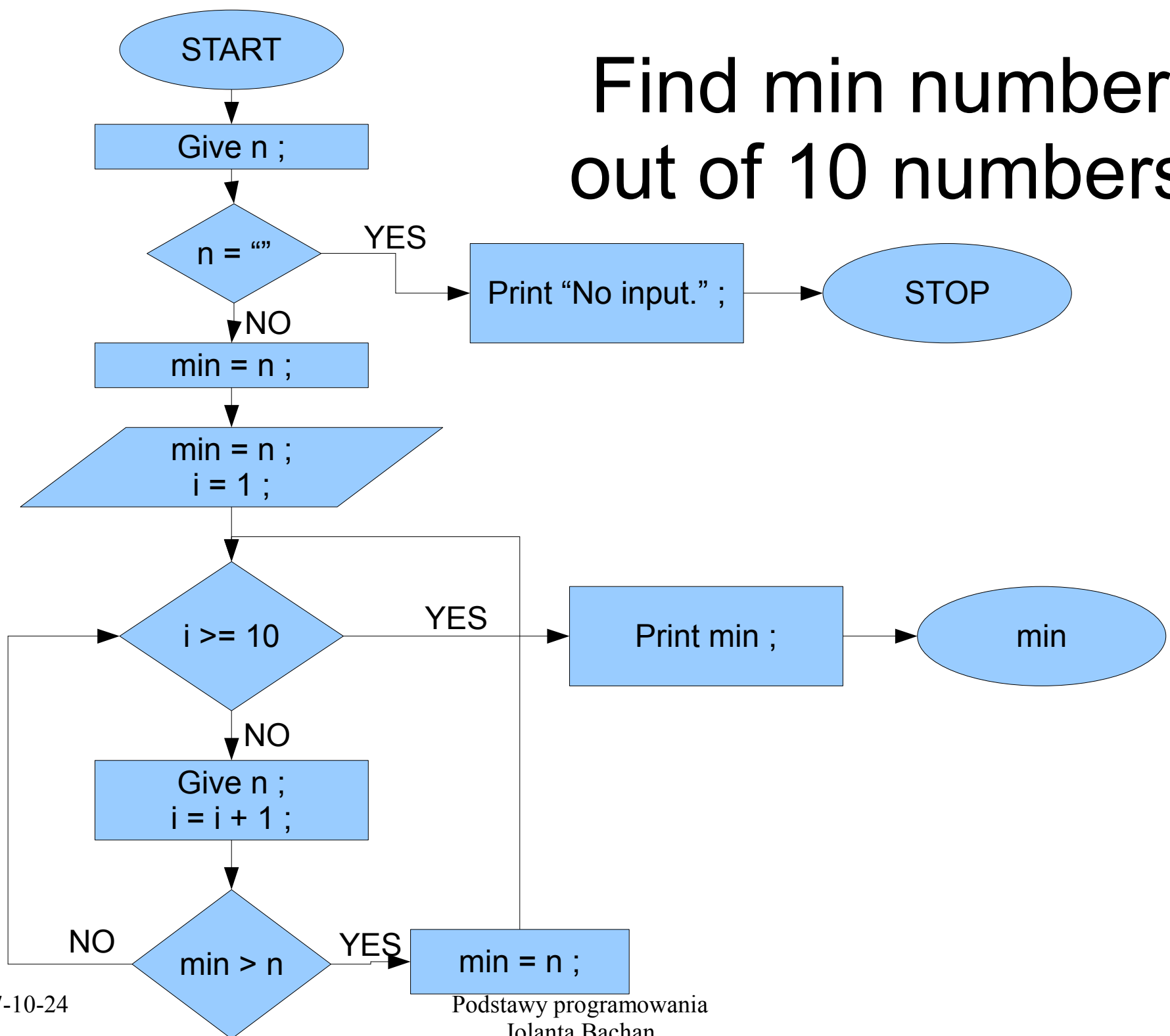
# The sum of 5 numbers



# Find max number out of 5 numbers



# Find min number out of 10 numbers



# Homework

- Write an algorithm which counts an average of 7 numbers.

# Homework

- Write an algorithm which counts an average of 7 numbers.
- Now – Implement the algorithm in Perl

See you in two weeks!