

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

# Functions

TEAM INFDEV

Hogeschool Rotterdam  
Rotterdam, Netherlands

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

# Introduction

## Lecture topics

- So far we have shown how data representation can be abstracted away
- Building useful containers only once makes it possible to reuse their definition
- Many data structures (tuples, lists, maps, sets, etc.) become thus a new layer of abstraction

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

## Lecture topics

- Manipulating these data structures happens in user code
- Often, user code needs to perform operations that are similar to each other
- Similar operations should not require rewriting everything every time

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

# Problem discussion

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

## Introduction

- Consider many operations on lists
  - finding or removing a specific element in a container
  - computing the length of a list
  - removing all elements that satisfy a condition
  - ...

# Lenght of a list

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

```
1 cnt = 0
2 x = l
3 while not(x.IsEmpty):
4     cnt = cnt + 1
5     x = x.Tail
6 print("List l contains " + str(cnt) + " elements.")
```

## Introduction

# Lenght of a list

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

```
1 cnt = 0
2 x = l
3 while not(x.IsEmpty):
4     cnt = cnt + 1
5     x = x.Tail
6 print("List l contains " + str(cnt) + " elements.")
```

## Introduction

- What does l contain?
- What do we do with the values of the list?
- Do they even matter?



Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

## Introduction

- Suppose that we now have another list,  $k$
- We wish to know its length
- How do we do it?

# Lenght of a list

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

```
1 cnt = 0
2 x = k
3 while not(x.IsEmpty):
4     cnt = cnt + 1
5     x = x.Tail
6 print("List_k contains " + str(cnt) + " elements.")
```

## Introduction

```
1 cnt = 0
2 x = k
3 while not(x.IsEmpty):
4     cnt = cnt + 1
5     x = x.Tail
6 print("List_k contains " + str(cnt) + " elements.")
```

## Introduction

- Looks suspiciously like the previous code block
- Why?

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

# General idea

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

## Adding our own layers

- The goal of this lecture is to add a new layer of abstraction to our programs
- We wish to reuse **implementations**, not only data structures
- This layer of abstraction is called **functions**

# Adding our own layers

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

```
1 +-----+
2 | ...      |
3 +-----+
4 | Functions      |
5 +-----+
6 | data structures      |
7 +-----+
8 | if, for, while, variables |
9 +-----+
10 | (Python) runtime      |
11 +-----+
12 ...
```

## Description

- A function is a collection of instructions and variables
- Some instructions and variables are fixed inside its **body**
- Other instructions and variables come from outside the function, and thus are not fixed; these are called **parameters** of the function
- We try to strike the right balance between flexibility and work done
- The function returns a final result that can be recovered by the code that uses the function

# Blueprint of a function (NOT ACTUAL PYTHON CODE!)

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

```
1  length of a list l:  
2      cnt = 0  
3      x = l  
4      while not(x.IsEmpty):  
5          cnt = cnt + 1  
6          x = x.Tail  
7      return cnt as the final result
```

## Description



# Blueprint of a function (NOT ACTUAL PYTHON CODE!)

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

```
1 length of a list l:  
2     cnt = 0  
3     x = l  
4     while not(x.IsEmpty):  
5         cnt = cnt + 1  
6         x = x.Tail  
7     return cnt as the final result
```

## Description

- `length` is the **function name**
- `l` is the only **parameter**
- Lines 2 through 6 are **fixed**
- `cnt` is the **final result**

## Using the function

- Code that needs the length of a function can now simply invoke function `length`
- The resulting code will simply be `l_len = length(l)`
- `l_len` will be assigned with the value returned by the function

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

# Technical details

## Introduction

- A function can be defined in Python quite easily
- The syntax is:
  - `def <<name>>(<<parameters>>):a`
    - `body`
    - `return <<result>>`
- Inside a function we can put whatever instructions we need
  - `if`
  - `for`
  - `...`

---

<sup>a</sup>Parameters might be none, thus we can write simply `()`

<sup>b</sup>Multiple parameters are separated by a comma, thus  
`(<<p1>>, <<p2>>, ..., <<pn>>)`

## Using the function

- After we declare a function, we can use it
- The syntax is quite simple
  - `<<name>>(<<parameters>>)` to just call the function and ignore the result
  - `<<v>> = <<name>>(<<parameters>>)` to call the function and assign the result to the `<<v>>` variable
- After calling the function, we enter the local environment of the function
- Variables, the PC, etc. are separate from those of the calling site

# Runtime example

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

S

PC
9

H


```
1 def length(l):  
2     cnt = 0  
3     x = l  
4     while not(x.IsEmpty):  
5         cnt = cnt + 1  
6         x = x.Tail  
7     return cnt  
8  
9 print(length(Node(10, Empty)))
```

# Runtime example

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

S	PC
	9

H	

```

1 def length(l):
2     cnt = 0
3     x = l
4     while not(x.IsEmpty):
5         cnt = cnt + 1
6         x = x.Tail
7     return cnt
8
9 print(length(Node(10, Empty)))

```

S	PC	length	PC	l
	9	nil	2	ref(1)

H	0	1
	[l ↦ True]	[l ↦ False; V ↦ 10; T ↦ ref(0)]

# Runtime example

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

S

PC	length	PC	I
9	nil	2	ref(1)

H

0	1
$[I \mapsto \text{True}]$	$[I \mapsto \text{False}; V \mapsto 10; T \mapsto \text{ref}(0)]$

```
1 def length(l):  
2   cnt = 0  
3   x = l  
4   while not(x.IsEmpty):  
5     cnt = cnt + 1  
6     x = x.Tail  
7   return cnt  
8  
9 print(length(Node(10, Empty)))
```



# Runtime example

## Functions

### TEAM INFDEV

## Introduction

## Problem discussion

## General idea

## Technical details

## Assignments

## Conclusion

S	PC	length	PC	I
	9	nil	2	ref(1)

H	0	1
	[I ↦ True]	[I ↦ False; V ↦ 10; T ↦ ref(0)]

```

1 def length(l):
2   cnt = 0
3   x = l
4   while not(x.IsEmpty):
5     cnt = cnt + 1
6     x = x.Tail
7   return cnt
8
9 print(length(Node(10, Empty)))

```

S	PC	length	PC	I	cnt
	9	nil	3	ref(1)	0

H	0	1
	[I ↦ True]	[I ↦ False; V ↦ 10; T ↦ ref(0)]

# Runtime example

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

S

PC	push	PC	I	cnt
9	length	3	ref(1)	0

H

0	1
$[I \mapsto \text{True}]$	$[I \mapsto \text{False}; V \mapsto 10; T \mapsto \text{ref}(0)]$

```

1 def length(l):
2   cnt = 0
3   x = l
4   while not(x.IsEmpty):
5     cnt = cnt + 1
6     x = x.Tail
7   return cnt
8
9 print(length(Node(10, Empty)))

```

# Runtime example

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

S	PC	push	PC	I	cnt
	9	length	3	ref(1)	0

H	0	1
	[I $\mapsto$ True]	[I $\mapsto$ False; V $\mapsto$ 10; T $\mapsto$ ref(0)]

```

1 def length(l):
2   cnt = 0
3   x = l
4   while not(x.IsEmpty):
5     cnt = cnt + 1
6     x = x.Tail
7   return cnt
8
9 print(length(Node(10, Empty)))

```

S	PC	length	PC	I	cnt	x
	9	nil	4	ref(1)	0	ref(1)

H	0	1
	[I $\mapsto$ True]	[I $\mapsto$ False; V $\mapsto$ 10; T $\mapsto$ ref(0)]

# Runtime example

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

*After a few steps...*

S	PC	length	PC	l	cnt	x
	9	nil	7	ref(1)	1	ref(0)

H	0	1
	[l $\mapsto$ True]	[l $\mapsto$ False; V $\mapsto$ 10; T $\mapsto$ ref(0)]

```

1 def length(l):
2   cnt = 0
3   x = l
4   while not(x.IsEmpty):
5     cnt = cnt + 1
6     x = x.Tail
7   return cnt
8
9 print(length(Node(10, Empty)))

```

# Runtime example

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

*After a few steps...*

S	PC	length	PC	l	cnt	x
	9	nil	7	ref(1)	1	ref(0)

  

H	0	1
	[l ↦ True]	[l ↦ False; V ↦ 10; T ↦ ref(0)]

```

1 def length(l):
2     cnt = 0
3     x = l
4     while not(x.IsEmpty):
5         cnt = cnt + 1
6         x = x.Tail
7     return cnt
8
9 print(length(Node(10, Empty)))

```

**Do we still need all the local variables of the function?**

# Runtime example

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

*After a few steps...*

S	PC	length	PC	l	cnt	x
	9	nil	7	ref(1)	1	ref(0)

  

H	0	1
	[l ↦ True]	[l ↦ False; V ↦ 10; T ↦ ref(0)]

```

1 def length(l):
2     cnt = 0
3     x = l
4     while not(x.IsEmpty):
5         cnt = cnt + 1
6         x = x.Tail
7     return cnt
8
9 print(length(Node(10, Empty)))

```

**Do we still need all the local variables of the function?**  
**Where do we put the result?**

# Runtime example

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

*After a few steps...*

S	PC	length	PC	l	cnt	x
	9	nil	7	ref(1)	1	ref(0)

H	0	1
	[l ↦ True]	[l ↦ False; V ↦ 10; T ↦ ref(0)]

```

1 def length(l):
2     cnt = 0
3     x = l
4     while not(x.IsEmpty):
5         cnt = cnt + 1
6         x = x.Tail
7     return cnt
8
9 print(length(Node(10, Empty)))

```

**Do we still need all the local variables of the function?  
Where do we put the result?**

S	PC	length
	9	1

H	0	1
	[l ↦ True]	[l ↦ False; V ↦ 10; T ↦ ref(0)]

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

## Syntax and semantics

- We will now describe how Python functions work precisely
- This is a **fundamental** bit of knowledge that determines if you really do learn how to program or not
- This **absolutely requires** a lot of focus to get
- Please panic a bit on the inside



## Subtleties that make functions “fun” to use

- About variables
  - Variables and parameters inside a function have precise **scope** (visibility)
  - Primitive values given as parameters can be **changed only locally** to the function
  - References given as parameters can be **permanently changed** from within the function
  - Global variables defined outside the function may be **read but not changed** from within the function<sup>a</sup>
- About behaviour
  - A function may **call itself**, in a process known as **recursion**
  - A function may **get as parameters and return other functions**, in a process known as **higher order functions**

---

<sup>a</sup>Unless you use some tricks we strongly discourage

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

## Local and global variables (basics of scope)

- The parameters of a function are added to the list of accessible variables
- They are only visible from inside the function
- Global variables are also visible from inside the function

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

## Local and global variables (basics of scope)

- Every call to a function generates a new value of the stack memory  $S$
- This contains (private copy of) all local variables
- The heap memory  $H$  remains the same
- The original stack memory (the **global variables**) remains accessible, just read-only

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

## Local and global variables (basics of scope)

- Every call to a function also reserves some special locations in the stack
- The local PC of the function
- The local variables of the function
- The returned value when the function is done

# Locals and globals

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

```
1 x = 1
2
3 def f(z):
4     return x * z
5
6 print(f(10))
7 print(f(30))
8 x = 2
9 print(f(10))
```

## Local and global variables (basics of scope)

- `x` is a global variable, visible outside and inside the function
- `z` is a local variable, visible only inside the function

# Locals and globals

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

```
1 x = 1
2
3 def f(z):
4     return x * z
5
6 print(f(10))
7 print(f(30))
8 x = 2
9 print(f(10))
```

## Local and global variables (basics of scope)

- $x$  is a global variable, visible outside and inside the function
- $z$  is a local variable, visible only inside the function
- **What does this program print?**

# Locals and globals

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

```
1 x = 1
2
3 def f(z):
4     return x * z
5
6 print(f(10))
7 print(f(30))
8 x = 2
9 print(f(10))
```

## Local and global variables (basics of scope)

- $x$  is a global variable, visible outside and inside the function
- $z$  is a local variable, visible only inside the function
- **What does this program print?**
- 10, 30, 20

# Locals and globals

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

S

PC
1

H


```
1 x = 1
2
3 def f(z):
4     return x * z
5
6 print(f(10))
7 x = 2
8 print(f(10))
```



# Locals and globals

## Functions

### TEAM INFDEV

## Introduction

## Problem discussion

## General idea

## Technical details

## Assignments

## Conclusion

S

PC
1

H


```

1 x = 1
2
3 def f(z):
4     return x * z
5
6 print(f(10))
7 x = 2
8 print(f(10))

```

S

PC	x
6	1

H


# Locals and globals

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

S

PC	x
6	1

H


```
1 x = 1
2
3 def f(z):
4     return x * z
5
6 print(f(10))
7 x = 2
8 print(f(10))
```

# Locals and globals

## Functions

### TEAM INFDEV

## Introduction

## Problem discussion

## General idea

## Technical details

## Assignments

## Conclusion

S

PC	x
6	1

H


```

1 x = 1
2
3 def f(z):
4     return x * z
5
6 print(f(10))
7 x = 2
8 print(f(10))

```

S

PC	x	f	PC	z
6	1	nil	4	10

H


# Locals and globals

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

S

PC	x	f	PC	z
6	1	nil	4	10

H


```
1 x = 1
2
3 def f(z):
4     return x * z
5
6 print(f(10))
7 x = 2
8 print(f(10))
```

# Locals and globals

## Functions

### TEAM INFDEV

## Introduction

## Problem discussion

## General idea

## Technical details

## Assignments

## Conclusion

S

PC	x	f	PC	z
6	1	nil	4	10

H


```

1 x = 1
2
3 def f(z):
4     return x * z
5
6 print(f(10))
7 x = 2
8 print(f(10))

```

S

PC	x	f
7	1	10

H


# Locals and globals

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

S

PC	x	f
7	1	10

H


```
1 x = 1
2
3 def f(z):
4     return x * z
5
6 print(f(10))
7 x = 2
8 print(f(10))
```

# Locals and globals

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

S

PC	x	f
7	1	10

H


```
1 x = 1
2
3 def f(z):
4     return x * z
5
6 print(f(10))
7 x = 2
8 print(f(10))
```

S

PC	x
8	2

H


# Locals and globals

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

S

PC	x	f	PC	z
8	2	nil	4	10

H


```
1 x = 1
2
3 def f(z):
4     return x * z
5
6 print(f(10))
7 x = 2
8 print(f(10))
```



# Locals and globals

## Functions

### TEAM INFDEV

## Introduction

## Problem discussion

## General idea

## Technical details

## Assignments

## Conclusion

S

PC	x	f	PC	z
8	2	nil	4	10

H


```

1 x = 1
2
3 def f(z):
4     return x * z
5
6 print(f(10))
7 x = 2
8 print(f(10))

```

S

PC	x	f
8	2	20

H


# Locals and globals

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

```
1 x = 1
2
3 def f(z):
4     return x * z
5
6 print(f(10))
7 x = 2
8 print(f(10))
9 print(z)
```

## Local and global variables (basics of scope)

- `x` is a global variable, visible outside and inside the function
- `z` is a local variable, visible only inside the function

# Locals and globals

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

```
1 x = 1
2
3 def f(z):
4     return x * z
5
6 print(f(10))
7 x = 2
8 print(f(10))
9 print(z)
```

## Local and global variables (basics of scope)

- `x` is a global variable, visible outside and inside the function
- `z` is a local variable, visible only inside the function
- **What does this program do?**

# Locals and globals

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

```
1 x = 1
2
3 def f(z):
4     return x * z
5
6 print(f(10))
7 x = 2
8 print(f(10))
9 print(z)
```

## Local and global variables (basics of scope)

- `x` is a global variable, visible outside and inside the function
- `z` is a local variable, visible only inside the function
- **What does this program do?**
- Crash with `NameError: name 'z' is not defined`

```
1 def f(z):  
2     z = z + 1  
3     return z * 2  
4  
5 print(f(10))  
6 print(f(30))
```

## Local and global variables (basics of scope)

- `z` is a local variable, visible only inside the function

```
1 def f(z):  
2     z = z + 1  
3     return z * 2  
4  
5 print(f(10))  
6 print(f(30))
```

## Local and global variables (basics of scope)

- `z` is a local variable, visible only inside the function
- **What does this program print?**

```
1 def f(z):  
2     z = z + 1  
3     return z * 2  
4  
5 print(f(10))  
6 print(f(30))
```

## Local and global variables (basics of scope)

- $z$  is a local variable, visible only inside the function
- **What does this program print?**
- 22, 62

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

## Shadowing

- The parameters of a function have priority over globals
- They supersede global variables of the same name



```
1 x = 1
2
3 def f(x):
4     return x * 2
5
6 print(f(10))
7 print(f(20))
```

## Shadowing

- `x` is a global variable, potentially visible inside the function
- `x` is also a local variable of the function, which has priority over the global `x`

```
1 x = 1
2
3 def f(x):
4     return x * 2
5
6 print(f(10))
7 print(f(20))
```

## Shadowing

- `x` is a global variable, potentially visible inside the function
- `x` is also a local variable of the function, which has priority over the global `x`
- **What does this program print?**

```
1 x = 1
2
3 def f(x):
4     return x * 2
5
6 print(f(10))
7 print(f(20))
```

## Shadowing

- $x$  is a global variable, potentially visible inside the function
- $x$  is also a local variable of the function, which has priority over the global  $x$
- **What does this program print?**
- 20, 40

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

S

PC	x
6	1

H


```
1 x = 1
2
3 def f(x):
4     return x * 2
5
6 print(f(10))
7 print(f(20))
```

# Shadowing

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

S

PC	x
6	1

H


```

1 x = 1
2
3 def f(x):
4     return x * 2
5
6 print(f(10))
7 print(f(20))

```

S

PC	x	f	PC	x
6	1	nil	4	10

H


S

PC	x	f	PC	x
6	1	nil	4	10

H


```
1 x = 1
2
3 def f(x):
4     return x * 2
5
6 print(f(10))
7 print(f(20))
```

# Shadowing

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

S

PC	x	f	PC	x
6	1	nil	4	10

H


```

1 x = 1
2
3 def f(x):
4     return x * 2
5
6 print(f(10))
7 print(f(20))

```

S

PC	x	f
7	1	20

H


S

PC	x	f
7	1	20

H


```
1 x = 1
2
3 def f(x):
4     return x * 2
5
6 print(f(10))
7 print(f(20))
```



# Shadowing

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

S

PC	x	f
7	1	20

H


```

1 x = 1
2
3 def f(x):
4     return x * 2
5
6 print(f(10))
7 print(f(20))

```

S

PC	x	f	PC	x
7	1	nil	4	20

H


S

PC	x	f	PC	x
7	1	nil	4	20

H


```
1 x = 1
2
3 def f(x):
4     return x * 2
5
6 print(f(10))
7 print(f(20))
```

S

PC	x	f	PC	x
7	1	nil	4	20

H


```

1 x = 1
2
3 def f(x):
4     return x * 2
5
6 print(f(10))
7 print(f(20))

```

S

PC	x	f
8	1	40

H


Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

## Recursion

- (Recursive) functions are all functions that call themselves in their bodies
- This is based on the principle of induction and in general a very powerful technique
- This leads to a compacter and often more easily correct representation
  - Code is not easier to read, especially to the untrained eye

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

## Recursion

- Remember that calling a function creates a new instance of stack memory
- Recursive functions do this a lot
- Each recursive call has its own environment

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

```
1 def length(l):  
2     if l.IsEmpty:  
3         return 0  
4     else:  
5         return length(l.Tail) + 1
```

## Recursion

- How many 1's shall we have?

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

```
1 def length(l):  
2     if l.IsEmpty:  
3         return 0  
4     else:  
5         return length(l.Tail) + 1
```

## Recursion

- How many 1's shall we have?
- As many as the nodes of the initial value

S

PC
7

H


```
1 def length(l):  
2     if l.IsEmpty:  
3         return 0  
4     else:  
5         return length(l.Tail) + 1  
6  
7 print(length(Node(1, Node(2, Empty))))
```



# Recursion

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

S

PC
7

H


```

1 def length(l):
2     if l.IsEmpty:
3         return 0
4     else:
5         return length(l.Tail) + 1
6
7 print(length(Node(1, Node(2, Empty))))

```

S

PC	length	PC	l
7	nil	2	ref(2)

H

0	1	2
$[l \mapsto \text{True}]$	$[l \mapsto \text{False}; V \mapsto 2; T \mapsto \text{ref}(0)]$	$[l \mapsto \text{False}; V \mapsto 1; T \mapsto \text{ref}(1)]$

# Recursion

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

S

PC	length	PC	I
7	nil	2	ref(2)

H

0	1	2
[I $\mapsto$ True]	[I $\mapsto$ False; V $\mapsto$ 2; T $\mapsto$ ref(0)]	[I $\mapsto$ False; V $\mapsto$ 1; T $\mapsto$ ref(1)]

```

1 def length(l):
2     if l.IsEmpty:
3         return 0
4     else:
5         return length(l.Tail) + 1
6
7 print(length(Node(1, Node(2, Empty))))

```

# Recursion

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

S

PC	length	PC	I
7	nil	2	ref(2)

H

0	1	2
$[I \mapsto \text{True}]$	$[I \mapsto \text{False}; V \mapsto 2; T \mapsto \text{ref}(0)]$	$[I \mapsto \text{False}; V \mapsto 1; T \mapsto \text{ref}(1)]$

```

1 def length(l):
2   if l.IsEmpty:
3     return 0
4   else:
5     return length(l.Tail) + 1
6
7 print(length(Node(1, Node(2, Empty))))

```

S

PC	length	PC	I
7	nil	5	ref(2)

H

0	1	2
$[I \mapsto \text{True}]$	$[I \mapsto \text{False}; V \mapsto 2; T \mapsto \text{ref}(0)]$	$[I \mapsto \text{False}; V \mapsto 1; T \mapsto \text{ref}(1)]$

# Recursion

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

S

PC	length	PC	I
7	nil	5	ref(2)

H

0	1	2
$[I \mapsto \text{True}]$	$[I \mapsto \text{False}; V \mapsto 2; T \mapsto \text{ref}(0)]$	$[I \mapsto \text{False}; V \mapsto 1; T \mapsto \text{ref}(1)]$

```

1 def length(l):
2   if l.IsEmpty:
3     return 0
4   else:
5     return length(l.Tail) + 1
6
7 print(length(Node(1, Node(2, Empty))))

```

# Recursion

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

S

PC	length	PC	I
7	nil	5	ref(2)

H

0	1	2
$[I \mapsto \text{True}]$	$[I \mapsto \text{False}; V \mapsto 2; T \mapsto \text{ref}(0)]$	$[I \mapsto \text{False}; V \mapsto 1; T \mapsto \text{ref}(1)]$

```

1 def length(l):
2   if l.IsEmpty:
3     return 0
4   else:
5     return length(l.Tail) + 1
6
7 print(length(Node(1, Node(2, Empty))))

```

S

PC	length	PC	I	length	PC	I
7	nil	5	ref(2)	nil	2	ref(1)

H

0	1	2
$[I \mapsto \text{True}]$	$[I \mapsto \text{False}; V \mapsto 2; T \mapsto \text{ref}(0)]$	$[I \mapsto \text{False}; V \mapsto 1; T \mapsto \text{ref}(1)]$

S

PC	length	PC	l	length	PC	l
7	nil	5	ref(2)	nil	2	ref(1)

H

0	1	2
[l $\mapsto$ True]	[l $\mapsto$ False; V $\mapsto$ 2; T $\mapsto$ ref(0)]	[l $\mapsto$ False; V $\mapsto$ 1; T $\mapsto$ ref(1)]

```

1 def length(l):
2     if l.IsEmpty:
3         return 0
4     else:
5         return length(l.Tail) + 1
6
7 print(length(Node(1, Node(2, Empty))))

```

S	PC	length	PC	l	length	PC	l
	7	nil	5	ref(2)	nil	2	ref(1)

H	0	1	2
	[l $\mapsto$ True]	[l $\mapsto$ False; V $\mapsto$ 2; T $\mapsto$ ref(0)]	[l $\mapsto$ False; V $\mapsto$ 1; T $\mapsto$ ref(1)]

```

1 def length(l):
2     if l.IsEmpty:
3         return 0
4     else:
5         return length(l.Tail) + 1
6
7 print(length(Node(1, Node(2, Empty))))

```

S	PC	length	PC	l	length	PC	l
	7	nil	5	ref(2)	nil	5	ref(1)

H	0	1	2
	[l $\mapsto$ True]	[l $\mapsto$ False; V $\mapsto$ 2; T $\mapsto$ ref(0)]	[l $\mapsto$ False; V $\mapsto$ 1; T $\mapsto$ ref(1)]

S

PC	length	PC	l	length	PC	l	length	PC	l
7	nil	5	ref(2)	nil	5	ref(1)	nil	2	ref(0)

H

0	1	2
[l $\mapsto$ True]	[l $\mapsto$ False; V $\mapsto$ 2; T $\mapsto$ ref(0)]	[l $\mapsto$ False; V $\mapsto$ 1; T $\mapsto$ ref(1)]

```

1 def length(l):
2     if l.IsEmpty:
3         return 0
4     else:
5         return length(l.Tail) + 1
6
7 print(length(Node(1, Node(2, Empty))))

```



S	PC	length	PC	l	length	PC	l	length	PC	l
	7	nil	5	ref(2)	nil	5	ref(1)	nil	2	ref(0)

H	0	1	2
	[l $\mapsto$ True]	[l $\mapsto$ False; V $\mapsto$ 2; T $\mapsto$ ref(0)]	[l $\mapsto$ False; V $\mapsto$ 1; T $\mapsto$ ref(1)]

```

1 def length(l):
2     if l.IsEmpty:
3         return 0
4     else:
5         return length(l.Tail) + 1
6
7 print(length(Node(1, Node(2, Empty))))

```

S	PC	length	PC	l	length	PC	l	length	PC	l
	7	nil	5	ref(2)	nil	5	ref(1)	nil	3	ref(0)

H	0	1	2
	[l $\mapsto$ True]	[l $\mapsto$ False; V $\mapsto$ 2; T $\mapsto$ ref(0)]	[l $\mapsto$ False; V $\mapsto$ 1; T $\mapsto$ ref(1)]

S

PC	length	PC	l	length	PC	l	length	PC	l
7	nil	5	ref(2)	nil	5	ref(1)	nil	3	ref(0)

H

0	1	2
[l $\mapsto$ True]	[l $\mapsto$ False; V $\mapsto$ 2; T $\mapsto$ ref(0)]	[l $\mapsto$ False; V $\mapsto$ 1; T $\mapsto$ ref(1)]

```

1 def length(l):
2     if l.IsEmpty:
3         return 0
4     else:
5         return length(l.Tail) + 1
6
7 print(length(Node(1, Node(2, Empty))))

```

S

PC	length	PC	l	length	PC	l	length	PC	l
7	nil	5	ref(2)	nil	5	ref(1)	nil	3	ref(0)

H

0	1	2
$[l \mapsto \text{True}]$	$[l \mapsto \text{False}; V \mapsto 2; T \mapsto \text{ref}(0)]$	$[l \mapsto \text{False}; V \mapsto 1; T \mapsto \text{ref}(1)]$

```

1 def length(l):
2   if l.IsEmpty:
3     return 0
4   else:
5     return length(l.Tail) + 1
6
7 print(length(Node(1, Node(2, Empty))))

```

S

PC	length	PC	l	length	PC	l	length
7	nil	5	ref(2)	nil	5	ref(1)	0

H

0	1	2
$[l \mapsto \text{True}]$	$[l \mapsto \text{False}; V \mapsto 2; T \mapsto \text{ref}(0)]$	$[l \mapsto \text{False}; V \mapsto 1; T \mapsto \text{ref}(1)]$

S

PC	length	PC	l	length	PC	l	length
7	nil	5	ref(2)	nil	5	ref(1)	0

H

0	1	2
[l $\mapsto$ True]	[l $\mapsto$ False; V $\mapsto$ 2; T $\mapsto$ ref(0)]	[l $\mapsto$ False; V $\mapsto$ 1; T $\mapsto$ ref(1)]

```

1 def length(l):
2     if l.IsEmpty:
3         return 0
4     else:
5         return length(l.Tail) + 1
6
7 print(length(Node(1, Node(2, Empty))))

```

S	PC	length	PC	l	length	PC	l	length
	7	nil	5	ref(2)	nil	5	ref(1)	0

H	0	1	2
	[l $\mapsto$ True]	[l $\mapsto$ False; V $\mapsto$ 2; T $\mapsto$ ref(0)]	[l $\mapsto$ False; V $\mapsto$ 1; T $\mapsto$ ref(1)]

```

1 def length(l):
2     if l.IsEmpty:
3         return 0
4     else:
5         return length(l.Tail) + 1
6
7 print(length(Node(1, Node(2, Empty))))

```

S	PC	length	PC	l	length
	7	nil	5	ref(2)	0+1

H	0	1	2
	[l $\mapsto$ True]	[l $\mapsto$ False; V $\mapsto$ 2; T $\mapsto$ ref(0)]	[l $\mapsto$ False; V $\mapsto$ 1; T $\mapsto$ ref(1)]

S

PC	length	PC	l	length
7	nil	5	ref(2)	1

H

0	1	2
$[l \mapsto \text{True}]$	$[l \mapsto \text{False}; V \mapsto 2; T \mapsto \text{ref}(0)]$	$[l \mapsto \text{False}; V \mapsto 1; T \mapsto \text{ref}(1)]$

```

1 def length(l):
2   if l.IsEmpty:
3     return 0
4   else:
5     return length(l.Tail) + 1
6
7 print(length(Node(1, Node(2, Empty))))

```

S	PC	length	PC	l	length
	7	nil	5	ref(2)	1

H	0	1	2
	$[l \mapsto \text{True}]$	$[l \mapsto \text{False}; V \mapsto 2; T \mapsto \text{ref}(0)]$	$[l \mapsto \text{False}; V \mapsto 1; T \mapsto \text{ref}(1)]$

```

1 def length(l):
2   if l.IsEmpty:
3     return 0
4   else:
5     return length(l.Tail) + 1
6
7 print(length(Node(1, Node(2, Empty))))

```

S	PC	length
	7	2

H	0	1	2
	$[l \mapsto \text{True}]$	$[l \mapsto \text{False}; V \mapsto 2; T \mapsto \text{ref}(0)]$	$[l \mapsto \text{False}; V \mapsto 1; T \mapsto \text{ref}(1)]$

# Recursion

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

S

PC	length
7	2

H

0	1	2
$[I \mapsto \text{True}]$	$[I \mapsto \text{False}; V \mapsto 2; T \mapsto \text{ref}(0)]$	$[I \mapsto \text{False}; V \mapsto 1; T \mapsto \text{ref}(1)]$

```

1 def length(l):
2     if l.IsEmpty:
3         return 0
4     else:
5         return length(l.Tail) + 1
6
7 print(length(Node(1, Node(2, Empty))))

```



# Recursion

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

S

PC	length
7	2

H

0	1	2
$[I \mapsto \text{True}]$	$[I \mapsto \text{False}; V \mapsto 2; T \mapsto \text{ref}(0)]$	$[I \mapsto \text{False}; V \mapsto 1; T \mapsto \text{ref}(1)]$

```

1 def length(l):
2   if l.IsEmpty:
3     return 0
4   else:
5     return length(l.Tail) + 1
6
7 print(length(Node(1, Node(2, Empty))))

```

S

PC
8

H

0	1	2
$[I \mapsto \text{True}]$	$[I \mapsto \text{False}; V \mapsto 2; T \mapsto \text{ref}(0)]$	$[I \mapsto \text{False}; V \mapsto 1; T \mapsto \text{ref}(1)]$

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

# Assignments

## Build and test, on paper...

- A function `add` that increments all elements of a list by a fixed value:
  - `add(10, Node(1,Node(2,Node(3,Empty)))) -> Node(11,Node(12,Node(13,Empty)))`
- A function `filterEven` that removes all odd elements from a list:
  - `filterEven(Node(1,Node(2,Node(3,Empty)))) -> Node(2,Empty)`
- A function `sum` that adds all elements of a list:
  - `sum(Node(1,Node(2,Node(3,Empty)))) -> 6`

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

# Conclusion

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

## Lecture topics

- Often, user code needs to perform operations that are similar to each other
- Through the mechanism of function definition, we can recycle code
- Functions can encode algorithms in many way
  - Simple code abstractions to avoid repetition
  - Recursive problems

Functions

TEAM  
INFDEV

Introduction

Problem  
discussion

General idea

Technical  
details

Assignments

Conclusion

The best of luck, and thanks for the  
attention!