



DM536

Introduction to Programming

Peter Schneider-Kamp

petersk@imada.sdu.dk

<http://imada.sdu.dk/~petersk/DM536/>

COURSE ORGANIZATION

Course Elements

- Lectures Monday 10-12 (every week)
- Lectures Wednesday 12-14 (next 3 weeks)
- 4 sections:
 - ???: Mathematics-Economy (2nd year)
 - ???: Mathematics / Applied Mathematics / Physics (2nd year)
 - S7 & S17: Computer Science (1st year)
- Discussion sections (marked “E” in your schedule)
- Labs (marked “E” in your schedule)
- Exam = 2 practical projects

Course Goals

- **Solve problems by writing computer programs**
- To this end, you will learn
 - to view programming as a problem solving approach
 - principles of imperative & object-oriented programming
 - how to model, test, debug, and document programs
- Focus on general principles, **NOT** on the language Python

Practical Issues / Course Material

- You need an IMADA account (\neq SDU account)
- Regularly check the course home page:
 - <http://imada.sdu.dk/~petersk/DM536/>
 - Slides, weekly notes, **definite** schedule, additional notes
- Reading material:
 - Allen B. Downey: *Think Python*, Green Tea Press, 2012.
 - Available as PDF and HTML from:
<http://greenteapress.com/thinkpython/thinkpython.html>

Course Contract

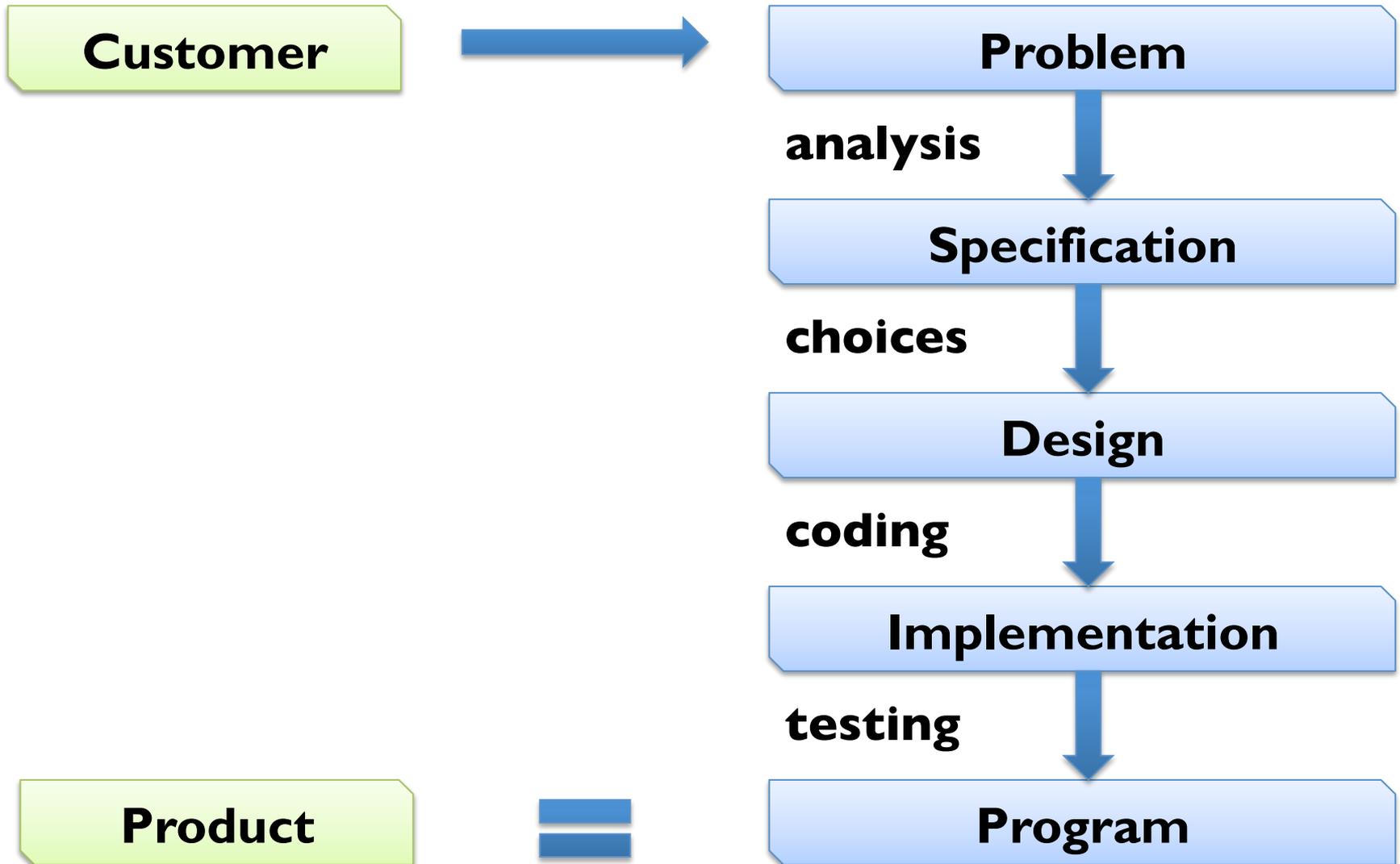
- I am offering you the following:
 1. I explain all needed concepts (as often as needed)
 2. I am available and always willing to help you
 3. I guide your learning by assigning exercises

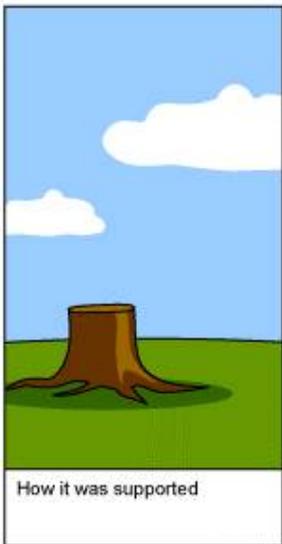
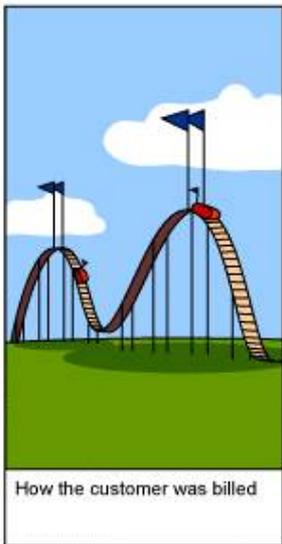
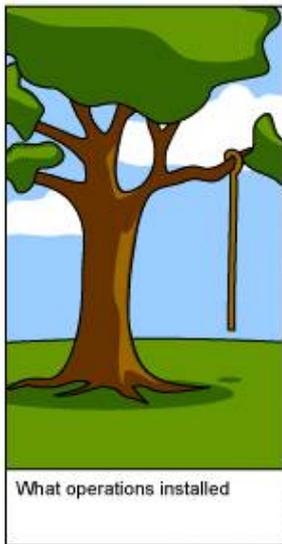
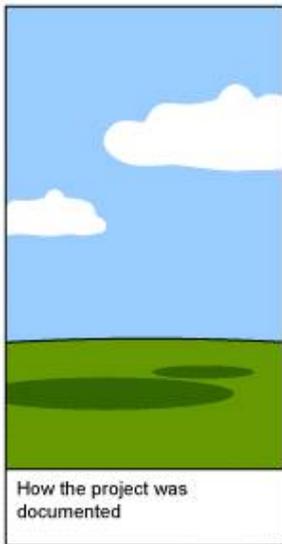
- From you I expect the following:
 1. You ask questions, when something is unclear
 2. You contact me (or a TA), when you need help
 3. You prepare for lectures and discussion sections

- You and I have the right and duty to call upon the contract!

PROGRAMMING

Programming as Problem Solving





Real Life “Programming”

Programming in a Nutshell

- Computers only have very limited abilities
- Computers are used to solve complex problems
- Programmers needed to break down complex problems into a sequence of simpler (sub-)problems
- program = sequence of simple instructions
- instructions = vocabulary of a programming language
- Programmers needed to express problems as sequence of instructions understandable to the computer

Simple Instructions

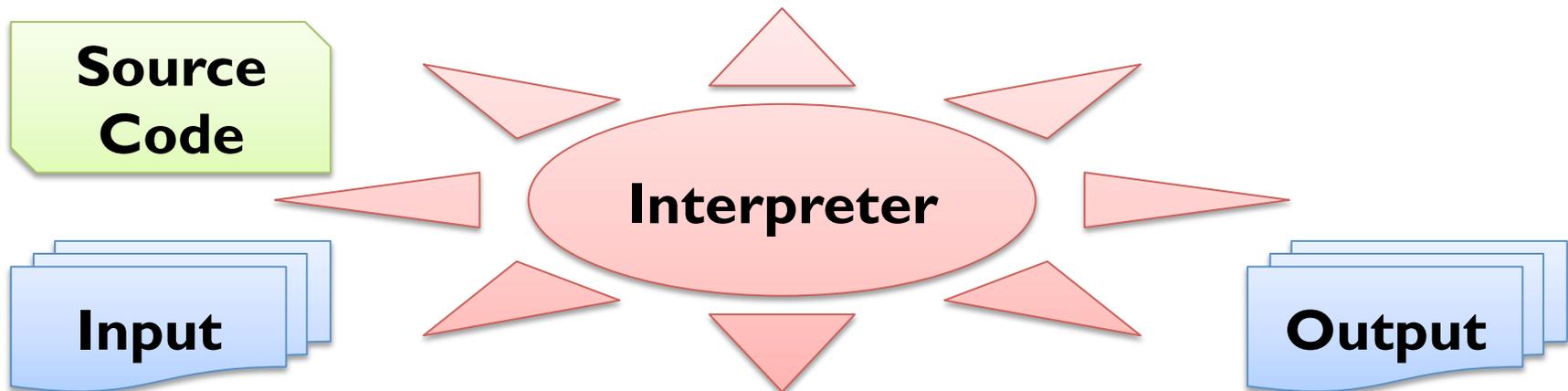
- Administrative: `from math import sqrt`
- Input:
`a = input()`
`b = input()`
- Arithmetic operations: `c = sqrt(a**2+b**2)`
- Output: `print "Result:", c`
- That is basically ALL a computer can do.

Combining Instructions

- Sequence: `<instr1>; <instr2>; <instr3>`
- Conditional Execution:
`if <cond>:`
 `<instr1>; <instr2>`
`else:`
 `<instr3>; <instr4>; <instr5>`
- Subprograms / Functions:
`def <function>(<argument>):`
 `<instr1>; <instr2>`
`<var> = <function>(<input>)`
- Repetition:
`while <cond>:`
 `<instr1>; <instr2>; <instr3>`

Executing Programs

- Program stored in a file (*source code* file)
- Instructions in this file executed top-to-bottom
- Interpreter executes each instruction



Debugging

- Any reasonably complex program contains errors
- Three types of errors (in Python)
 - Syntax Errors `a = input)(`
 - Runtime Errors `c = 42 / 0`
 - Semantic Errors `c = a**2+b**2`
- Debugging is finding out why an error occurred

VARIABLES, EXPRESSIONS & STATEMENTS

Values and Types

- Values = basic data objects 42 23.0 "Hello!"
- Types = classes of values integer float string

- Values can be printed:
 - `print <value>` `print "Hello!"`

- Types can be determined:
 - `type(<value>)` `type(23.0)`

- Values and types can be compared:
 - `<value> == <value>` `type(3) == type(3.0)`

Variables

- variable = name that refers to a value
- program state = mapping from variables to values
- values are *assigned* to variables using “=”:
 - `<var> = <value>` `b = 4`
- the value referred to by a variable can be printed:
 - `print <var>` `print b`
- the type of a variable is the type of the value it refers to:
 - `type(b) == type(4)`

Variable Names

- start with a letter (convention: a-z)
- contain letters a-z and A-Z, digits 0-9, and underscore “_”
- can be any such name except for 31 reserved names:

and	del	from	not	while
as	elif	global	or	with
assert	else	if	pass	yield
break	except	import	print	
class	exec	in	raise	
continue	finally	is	return	
def	for	lambda	try	

Multiple Assignment

- variables can be assigned to different values at different times:
 - Example: $x = 3$
 $x = 4$
 - Instructions are executed top-to bottom => x refers to 4
- be careful, e.g., when exchanging values serially:
 - Example: $x = y$
 $y = x$
 - later x and y refer to the same value
 - Solution 1 (new variable): $z = y; y = x; x = z$
 - Solution 2 (parallel assign.): $x, y = y, x$

Operators & Operands

- Operators represent computations: + * - / **
 - Example: $23+19$ day+month*30 $2**6-22$
- Addition “+”, Multiplication “*”, Subtraction “-” as usual
- Exponentiation “**”: $x**y$ means x^y
- Division “/” rounds down integers:
 - Example 1: $21/42$ has value 0, **NOT** 0.5
 - Example 2: $21.0/42$ has value 0.5
 - Example 3: $21/42.0$ has value 0.5

Expressions

- Expressions can be:

- Values: 42 23.0 "Hej med dig!"
- Variables: x y name|234
- built from operators: 19+23.0 x**2+y**2

- grammar rule:

- $\langle \text{expr} \rangle \Rightarrow \langle \text{value} \rangle$ |
- $\langle \text{var} \rangle$ |
- $\langle \text{expr} \rangle \langle \text{operator} \rangle \langle \text{expr} \rangle$ |
- $(\langle \text{expr} \rangle)$

- every expression has a value:
 - replace variables by their values
 - perform operations

Operator Precedence

- expressions are evaluated left-to-right
 - Example: $64 - 24 + 2 == 42$
- **BUT**: like in mathematics, “*” binds more strongly than “+”
 - Example: $2 + 8 * 5 == 42$
- parentheses have highest precedence: $64 - (24 + 2) == 38$
- **PEMDAS** rule:
 - Parentheses “(<expr>)”
 - Exponentiation “**”
 - Multiplication “*” and Division “/”
 - Addition “+” and Subtraction “-”

String Operations

- Addition “+” works on strings:
 - Example 1: `print "Hello w" + "orld!"`
 - Example 2: `print "4" + "2"`
- Multiplication “*” works on strings, if 2nd operands is integer:
 - Example: `print "Hej!" * 10`
- Subtraction “-”, Division “/”, and Exponentiation “**” do **NOT** work on strings