University of Zurich[UZH]

Department of Political Science

Lecture 1: Information Coding & Data Structures

Seminar ‘Foundations of Data Science’

Prof. Dr. Karsten Donnay, Assistant: Marcel Blum

Overview of this Session

- Administrative Aspects
- Motivation for the Course
- Information Coding
- Data Structures

University of Zurich[UZH]

Department of Political Science

Administrative Aspects

Formalities

- MA-level Seminar “Foundations of Data Science for Social Scientists” (no. 615d539a)
  - Lecture Team:
    - Karsten Donnay (donnay@ipz.uzh.ch)
    - Marcel Blum (marcel.blum@uzh.ch)
  - Course dates: 14.06.2021 – 18.06.2021
    - Lecture: 10:15 – 12:00
    - Exercise: 14:00 – 15:45
  - Online teaching only:
    - Zoom: https://uzh.zoom.us/j/91462284423 (Password: 810882)
    - OLAT: https://lms.uzh.ch/url/RepositoryEntry/16964059152

Course Objectives

- Develop a good understanding of technical and conceptual foundations of data science approaches in the social sciences.
- Learn to apply them in relevant research settings while adhering to best practices and standards of data quality and reproducibility.
- Familiarize yourselves with relevant tools and approaches and learn to apply them to your own research questions.

Course Assessment

- Based on written exercises and assignments
  - Total of four shorter exercises with assignments
    - Given out during the exercises on Mon. through Thu.
    - Due the **next day** before the next exercise session, i.e., **before 14:00**
    - 5 points per assignment for a total of 20 points
  - Longer final project (coding and report)
    - Assigned on the last day of the class, i.e., Fri. Jun. 18
    - Due the **following week** on Fri. **Jun. 25 at 23:59**
    - 30 points for the final project
  - Final grade:
    - 40% (20 points) from exercises, 60% (30 points) from final project

Course Structure

– **Lectures:**
  Technical and theoretical background on key data science concepts and overview of practical applications in the social sciences.

– **Exercises & Assignments:**
  Practical walk-through of a data science pipeline to provide you with examples, hands-on experience and an overview of best practices.

– **Final Project:**
  Applying those skills in a data science application of your own choice including
  - Data collection/import and cleaning
  - Processing and analysis
  - Simple research application and presentation of results

Assignments – Details

- Typically three individual tasks
  - Made available through GitHub and on OLAT
  - Hand-in prior to deadline through “Drop Box” function on OLAT
  - Feedback and grading also through “Drop Box”

- Submission format
  - Should be solved directly in the Markdown file (please submit as .Rmd)
  - We encourage you to talk to your fellow students and **collaborate** but everyone has to submit their **own file as solution**, i.e., no straight copy-paste
  - Grading out of 5 points for each assignment (total of 20 points for all assignments)

Final Project – Details

- Idea:
  - Work on a topic you are interested in or that is related to your research/thesis
  - Identify a concrete **research question** you are trying to address
  - Apply the lessons-learned from this class in the context of this question
- Format:
  - Full data science pipeline from initial data collection/processing to data wrangling, extracting relevant information, analysis and presentation of results
  - Should also be done in Markdown and doubles as your project report; emphasis is on coding but we also expect
    - Motivation for your research question (& short overview of related research)
    - Text narrative that leads through al parts of the data science pipeline
    - Discussion of results and potential shortcomings

Access to Course Material

- OLAT: https://lms.uzh.ch/url/RepositoryEntry/16964059152
  - Main course resource for
    - Course syllabus & announcements
    - Lecture slides & recordings
    - Exercises & assignments
    - Solutions for assignments (after respective deadline)
- GitHub: https://github.com/css-zurich/fds-2021
  - Quick access to course materials
    - Lecture slides
    - Exercises & assignments
    - .Rmd files etc.

Course Outline

- Part 1: Foundations
  - *Day 1: Mon. 14.06.2021*
    - 10:15 – 12:00 Lecture 1: Information Coding & Data Structures (SWITCHcast)
    - 14:00 – 15:45 Exercise Session 1 (Zoom)
  - *Day 2: Tue. 15.06.2021*
    - 10:15 – 12:00 Lecture 2: Programming & Algorithms (SWITCHcast)
    - 14:00 – 15:45 Exercise Session 2 (Zoom)
  - *Day 3: Wed. 16.06.2021*
    - 10:15 – 12:00 Lecture 3: Complexity & Efficiency (SWITCHcast)
    - 14:00 – 15:45 Exercise Session 3 (Zoom)

Course Outline

- Part 2: Applications
  - *Day 4: Thu. 17.06.2021*
    - 10:15 – 12:00 Lecture 4: Data Collection & Quality (SWITCHcast)
    - 14:00 – 15:45 Exercise Session 4 (Zoom)
  - *Day 5: Fri. 18.06.2021*
    - 10:15 – 12:00 Lecture 5: Research on Digital Media (SWITCHcast)
    - 14:00 – 15:45 Exercise Session 5 (Zoom)

Motivation for the Course

Motivation for the Course

- We are in the middle of a data “revolution”
  - Wealth of new data that is becoming available
    - Very large
    - Frequently updated
    - Diverse sources
  - Computational methods critically important to handle and analyze these data
    - Automatization
    - Robustness
    - Reproducibility

Motivation for the Course

– BUT lack of fundamental understanding of the “tools” we use
  - Often only a working knowledge of high-level software programming
    – SPSS
    – STATA
    – R (but primarily only for statistics)
  - Complex research problems require more advanced skills
    – Automatize data retrieval and handling
    – Design efficient algorithms
    – Develop or adapt software that implements specific methods

Computational Social Science or Data Science?

- **Definition:** Computational Social Science (CSS)
  Computational social science refers to the academic sub-disciplines concerned with **computational approaches to the social sciences**. This means that computers are used to model, simulate, and analyze social phenomena.
- Computational Social Science vs. Data Science
  - Data science is typically more narrowly concerned with extracting knowledge from data:
    - Machine learning
    - Data mining
    - Statistics
    - Predictive analytics
  - We are in this class interested in data science applications in the social sciences, i.e., we really are landing more on the side of Computational Social Science…

University of Zurich[UZH]

Department of Political Science

Information Coding

Units of Information

– Smallest physical unit is 1 **bit**: 0 or 1
  - memory: a capacitor or a capacitor and several transistors
  - hard disk: magnetization
– Smallest logical unit is 1 **byte**: 8 bit
  - processor can usually only address entire bytes
  - i.e., coding 1 bit of information “wastes” 7 bits but typically individual pieces of information we store already require more than one bit to code

Units of Information

- Machine word:
  - fixed-sized piece of data handled as a unit by the instruction set or the hardware of a processor
  - depends on the width of the processor bus
    - 8 bit (1 byte) for 8088
    - 16 bit (2 byte) for 80286
    - 32 bit (4 byte) for 80686
    - 64 bit (8 byte) in most of today's processors
    - 128 bit (16 byte) for example in graphic cards
  - processor is fastest at processing whole machine words

Number Systems

- Unary Numeral Systems:
  - every natural number is represented by a sequence of symbols
  - to represent number N, an arbitrary symbol representing 1 is repeated N times
- Positional Notation:
  - composed of a finite number of numeric symbols
  - represent arbitrary numbers through positioning of those numeric symbols

Number Systems

- Examples:
  - Roman numerals:
    - hybrid notation: partially unary, but also positional notation
    - III = 3, MMXVI = 2016, MCMLXXXIII = 1983
  - Arabic numerals
    - positional notation
    - only 10 different numerical digits
    - includes a representation for 0

Arabic Numerals

- Positional notation with base β
- Base β is also often referred to as radix
  - radix 10: decimal notation (0 – 9)
  - radix 2: binary notation (0, 1)
  - radix 8: octal notation (0 – 7)
  - radix 16: hexadecimal notation (0 – 9, A – F)
- Notation convention is to indicate the base, if not clearly known, as subscript
  - $7_{10} = 7_8 = 111_2$
  - $9_{10} = 11_8 = 1001_2$
  - $14_{10} = 16_8 = 1110_2$

Examples

Decimal notation

- $5648_{10} = 5 \cdot 10^3 + 6 \cdot 10^2 + 4 \cdot 10^1 + 8 \cdot 10^0$

Binary notation

- $21_{10} = 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 10101_2$

Octal notation

- $81_{10} = 1 \cdot 8^2 + 2 \cdot 8^1 + 1 \cdot 8^0 = 121_8$

Conversion Between Number Systems

– Conversion of octal to binary notation
  - every digit 0 – 7 can be represented exactly by 3 binary digits
  - allows for elegant and fast conversion digit by digit using blocks of 3 binary digits

Examples

- $123456_8 = 001\ 010\ 011\ 100\ 101\ 110_2$
- $4231_8 = 100\ 010\ 011\ 001_2$

Conversion Between Number Systems

– Conversion of hexadecimal to binary notation
  - every digit 0 – 9 and letters A – F can be represented by exactly 4 binary digits
  - allows for elegant and fast conversion digit by digit using blocks of 4 binary digits

Examples

- $23A7_{16} = 0010\ 0011\ 1010\ 0111_2$
- $B3D2_{16} = 1011\ 0011\ 1101\ 0010_2$

Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Binary	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

Negative Numbers

- Offset binary (or excess k) coding
  - move origin (0000) in the middle of the k digits
  - $z' = z - z^{k-1}$
  - fixed offset of all numbers, many values can not be used
  - simple binary arithmetic does not work any longer

decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
unsigned	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

decimal	-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
excess k	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

Negative Numbers

- Ones' Complement
  - leading bit codes the sign (0: +, 1: –)
  - $k-1$ digits to represent the absolute value of a given number
  - negation by inverting of the number
    - bit-wise flipping of 1s and 0s
  - two representations of 0:
    - 0000 0000
    - 1111 1111
  - addition requires "end-around carry"

Example

- $27_{10} = \underline{0}001\ 1011_2$
- $-27_{10} = \underline{1}110\ 0100_2$

Negative Numbers

- Two's Complement
  - leading bit codes the sign (0: +, 1: –)
  - negation by inverting of the number and adding 1
  - unique representation of 0: 0000 0000

Example

$27_{10} = 0001\ 1011_2$

$1110\ 0100_2$ (inverted)

$1110\ 0101_2 = -27_{10}$

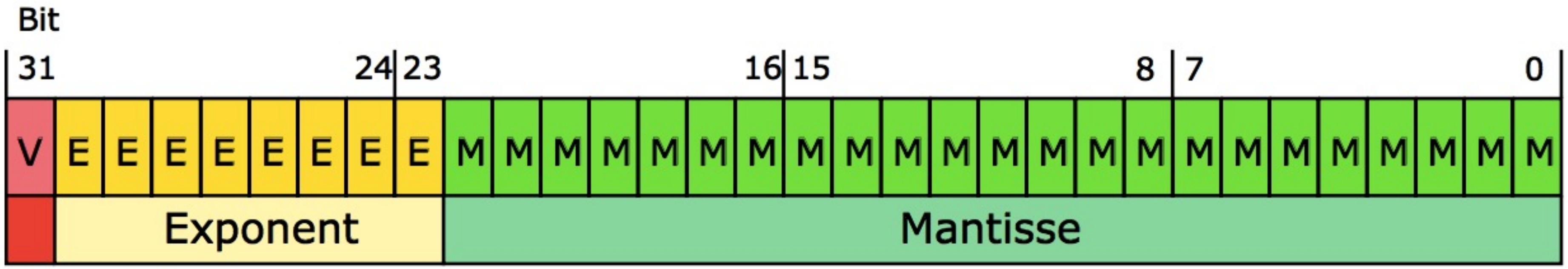
Real Numbers

- Fixed-point arithmetic
  - fixed assignment of k bits pre-decimal and decimal places
  - very inflexible
- Floating point
  - flexible division in sign (s), significand or mantissa (m) and exponent (e)
  - $z = s \cdot m \cdot \beta^e$

Real Numbers

- Floating point is standardized according to IEEE 754:
  - simple or double precision (32 or 64 bit)
  - 1 bit for the sign: $s = -1^S$
  - 23/52 bit significand with implicit 1: $m = 1.M$
    - leading 1 is not explicitly stored
    - significand and exponent are adjusted accordingly (normalized)
  - 8/11 bit exponent with bias: $\mathrm{e} = E - B$
    - bias permits sign less storage of negative exponents
    - $B = 127$ or $B = 1023$

IEEE 754



- Special representation for
  - 0: exponent and significand are both 0, i.e., there exists a positive and negative 0
  - ∞: exponent E only 1s, significand $M = 0$
- Value range:
  - simple precision (32 bit): $1.4 \cdot 10^{-45} \leq |z| \leq 3.4 \cdot 10^{38}$
  - double precision (64 bit): $4.9 \cdot 10^{-324} \leq |z| \leq 1.7 \cdot 10^{308}$

IEEE 754

Example

18.625 in single precision

- bias $B = 127 = 2^7 - 1$
- transform to binary notation:
  $18.625_{10} = 10010.101000\ldots_2$
- normalize:
  $10010.101000\ldots_2 = 1.0010101000\ldots \cdot 2^4$
- exponent $E = 4_{10} + 127_{10} = 131_{10} = 1000\ 0011_2$
- sign $S = 0$
- result:
  0|10000011|00101010000000000000000

Character Coding

- ASCII – Code: *American Standard Code for Information Interchange*
  - Standard since 1963, still in use today
  - initially 7 bit code for use in tele printers
    - 128 characters: 33 control characters, 95 printable characters

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Character Coding

– Extension of ASCII-Codes to 8 bit (1 byte)
  - Code pages or character sets
    – CP850, CP437, IS0-8859-1 and ISO-8859-15 with special characters for Western Europe

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL
8	PAD	HOP	BPH	NBH	IND	NEL	SSA	ESA	HTS	HTJ	VTS	PLD	PLU	RI	SS2	SS3
9	DCS	PU1	PU2	STS	CCH	MW	SPA	EPA	SOS	SGCI	SCI	CSI	ST	OSC	PM	APC
A	NBSP	¡	¢	£	€	¥	Š	§	š	©	ª	«	¬	SHY	®	¯
B	°	±	²	³	Ž	µ	¶	·	ž	¹	º	»	Œ	œ	Ÿ	¿
C	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Character Coding

- Unicode
  - Addresses problems with ASCII and code pages
    - not all possible characters can be represented, e.g., kanji
    - meaning of a sign is dependent on code page
  - Development of standardized Unicode
    - simultaneous representation of all possible characters
    - fixed translation table
    - version 1.0 in 1991 with 65’536 different characters
    - current version 13.0 (as of March 2020) defines 143’859 of 1’114’112 possible characters

Unicode

- Division of full code space into 17 planes with 65 536 characters
  - specification using U+ and at least 4 hexadecimal numbers
  - one code correspond to exactly one character, e.g.: U+00DF = β
- Plane 0, Basic Multilingual Plane (BMP)
  - currently used character sets, punctuation marks, control characters etc.
  - highly fragmented and mostly occupied
- Plane 1, Supplementary Multilingual Plane (SMP)
  - historical characters
  - domino and mahjong pieces
- Plane 2, Supplementary Ideographic Plane (SIP)
  - Chinese, Japanese and Korean characters

Representation of Unicode

- 3 bytes are necessary for all possible characters
  - not ideal for processing because it is not a machine word
  - “wasting” memory in many geographical regions, e.g. Europe
- Definition of Unicode Transformation Formats (UTF) as solution
  - UTF-16
    - 2 byte per characters, covers BMP
    - other areas are covered by combination of UTF-16 characters, still wastes memory
  - UTF-8
    - 1 byte per character, covers most important Western characters
    - additional characters can be covered by a combination of up to three UTF-8 characters
  - There also exists UTF-7 and UTF-32 but not as commonly used

Primitive Data Types

Type	Bits	Coding	Minimum	Maximum
boolean	1/8	truth value	false	true
byte	8	two's complement	-128	127
short	16	two's complement	-32 768	32 767
int	32	two's complement	-2 147 483 648	2 147 483 647
long	64	two's complement	-9 223 372 036 854 775 808	9 223 372 036 854 775 807
char	8-32	UTF-8 (standard)	0 (U+0000)	1 114 112 (U+0010FFFF)
float	32	IEEE 754	$\pm 1.4 \cdot 10^{-45}$	$\pm 3.4 \cdot 10^{38}$
double	64	IEEE 754	$\pm 4.9 \cdot 10^{-324}$	$\pm 1.7 \cdot 10^{308}$

Typing and Conversion

- R automatically types inputs
  - “Hello World” is automatically recognized as character string
  - 3.4 is automatically coded in floating point
    - R by default only has 64-bit double-precision
- Explicit type conversion is possible
  - Transform to numeric using as.numeric()
  - Transform to character using as.character()
    - Stores numbers as strings of characters
  - Transform to integer using as.integer()
    - Careful because simply “cuts of” decimal part

Strings

- Strings are immutable objects
  - Simply a sequence of individual characters
  - No standard “arithmetics” possible on strings
- R knows various methods for manipulation
  - Various “paste” commands to join strings
  - grep() (and similar commands) for pattern matching and replacement
    - Important together with regular expressions (RE) to match complex patterns in text
    - More on this in the second exercise, i.e., tomorrow afternoon
  - Important: “==” does **exact** string matching in R

Data Structures

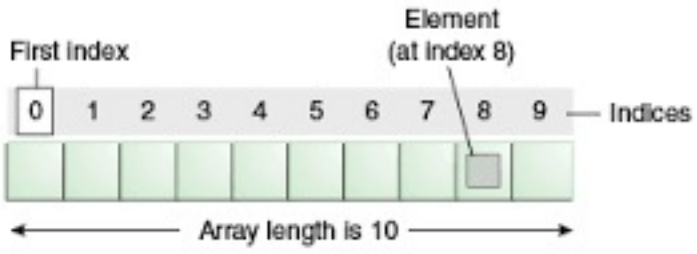
Data Structures

- **Definition:**
  Data structures are defined as data types together with operations defined on these data that enable and realize their access and management.
- We focus here on a few data structure with relevance for information storage
  - Data types typically vary by programming language
  - Primary data types used also depend on programming paradigm (more on this tomorrow)



Arrays

– **Definition:**
  An array is a systematic arrangement of similar objects, usually in rows and columns.



- We address/access elements using the **index**
- In most programming languages indices from from 0 to n-1 for array of length n
- In R, we use indices from 1 to n
  – e.g.: a = c(5, 4, 3, 2, 1), a[2] = 4

Lists

– **Definition:**
Lists are generic vectors containing other objects/data structures. This exact type of data structure is specific to R.

- The idea is to use lists in R for collections of different types of data, i.e., collections of numeric and character values
  - Important: Arrays in R are typed, i.e., c(1, “2”, 3) converts all numeric to character
  - If you want this kind of “mixed” collections you have to use lists
- Working with lists
  - [] gives you access to the whole list element
  - [[]] gives you access to the entries within that list element
  - list elements can have labels or no labels b = list(a = c(5, 4, 3, 2, 1)) or b = list(c(5, 4, 3, 2, 1))
    - If elements are labeled, you can access them with $, i.e., b$a, else with [] only

Data Frames

– **Definition:**
  Data frames is two dimensional data structure in R. It is a special case of a list which has each component of equal length, i.e., in other programming languages this is also often called a matrix or an nxm-array (i.e., n lines, m columns).

  - This data structure is the go-to solution for any kind of table structures in R, i.e., your usual spreadsheet data
  - It is relatively easy and fast to access and manipulate, also iteratively; you should be familiar with it already from your classes in statistics
  - Working with data frames
    - Accessing a given element requires specifying two indices, e.g. a[i,j] gives i-th row, j-th column of a
    - Accessing a whole row or column means leaving index j or i, respectively, empty
      - This is also often referred to as “slicing”

Other Data Structures

- There are many other important data structures, most of which have an implementation in R
  - All have specific applications in mind and are not suitable for every kind (or size) of data
  - Access and data manipulation might be feasible for some applications and not for others
  - In the following we take a look at one more important data structure that you will encounter quite often, the exercises touch upon a few others

Dictionaries

– **Definition:**
  Dictionaries are an abstract data type composed of a collection of (key, value) pairs, such that each possible key appears at most once in the collection.

- Generalized keys (not just indices):
  - Characters, words, names etc.
- Characteristics
  - Keys are not necessarily homogeneously distributed
    - Using an array spanning the whole range of values would waste memory
  - Use list of keys to index the dictionary
    - If no further conditions applied: sequential search
    - Worst case: have to search through ALL entries to find one

Dictionaries

- Hash Tables (intuition)
  - Dictionaries typically use hash tables to effectively index sparse key values
  - Examples of hash tables
    - student ID numbers
      - 6-digit number, i.e. 10^6 possible values
      - BUT only of a few 10,000s are used at the same time
      - can map all numbers onto those slots
    - books in the library
      - book titles have maybe ≈50 characters that means $\sim 26^{50}$ possible titles
      - BUT a library has “only” millions of books

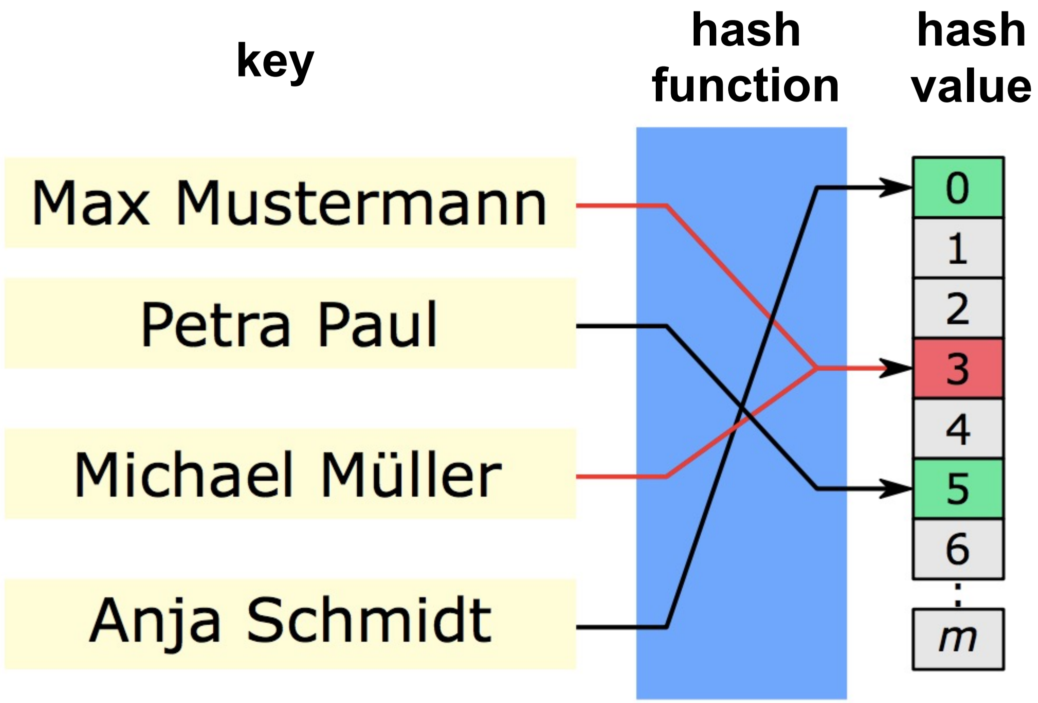
Dictionaries

– Hash Tables (formal)

- Also referred to as associative arrays and used also in databases, cryptography etc.
- **Definition: Hash function**
  A hash function $h: K \rightarrow \{0,1,\dots,m-1\}$ assigns to any given key k an index $0 \leq h(k) \leq m-1$.
  – Efficient mapping of indices but problem of so called **hash collisions**
    - Since we usually have $\mathrm{n} \gg m$ there is a very high likelihood that different keys get the same hash value
    - Requires special treatment that is dependent on how many hash values are already occupied

Hash Tables

– Example of hash collision



Hash Tables

- Hash function:
  - Central requirements
    - simple and fast to determine
    - balanced distribution to available indices to avoid collisions
    - fully deterministic calculation
    - (dynamically) adjustable to the number of free slots
  - Hash functions are usually based on natural numbers (with 0), i.e., $\mathrm{K} \subseteq N_0$
  - Usually direct calculation of the hash value of a given key (e.g., as a character string)

Hash Tables

– Examples for hash function:

- Modulo operation
  – $\mathrm{h(k)} = k \; mod \; m$ gives value in range 0 to m-1 and balanced distribution
  – Best choice for avoiding hash collisions are prime numbers

- Multiplicative hashing
  – Multiplication of an integer with an irrational number Θ and cutting of the integer part, then multiplication with m: $h(k) = \lfloor m(k \cdot \Theta - \lfloor k \cdot \Theta \rfloor) \rfloor$
  – Best results for the golden ratio $\phi^{-1} = \frac{\sqrt{5}-1}{2} \approx 0.6180339887 \ldots$

Up Next

- Exercise (this afternoon)
  - Data Science pipeline
  - Git & GitHub
  - First exercises in R with our working case
- Next lecture (tomorrow morning)
  - Programming
  - Algorithms