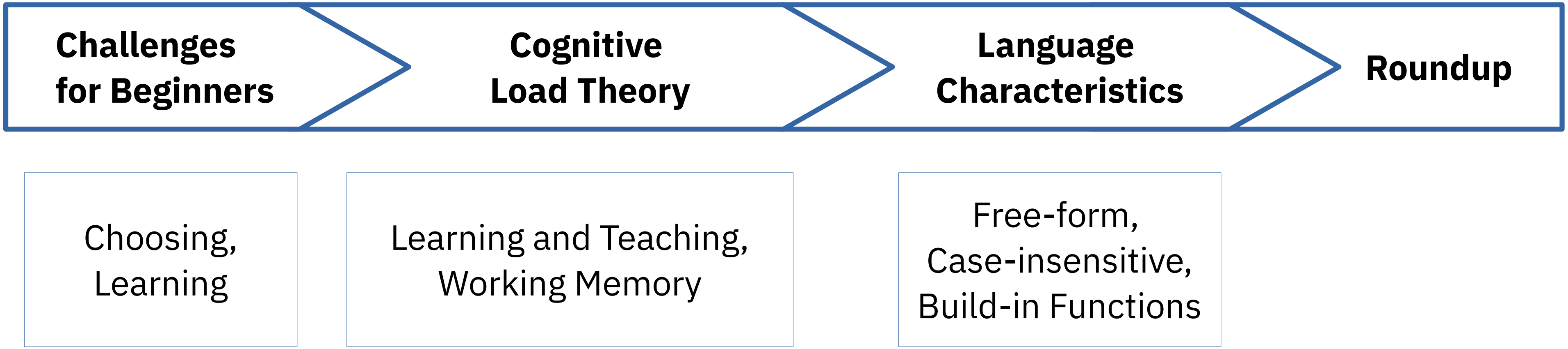
Department of Information Systems
and Operations Management

REXX from a Cognitive Load Perspective

The 34th International Rexx Symposium


Challenges for Beginners

Cognitive Load Theory

Language Characteristics

Roundup

Choosing, Learning

Learning and Teaching, Working Memory

Free-form, Case-insensitive, Build-in Functions

Till Winkler

Challenges for Beginners

Choosing a Language is Difficult!

- **Amount:** Wikipedia lists 691 different programming languages
  (“List of programming languages,” 2023)
- **Preference:** “Experts” usually state preferences
- **Popularity:** Frequency of online searches (PYPL, 2023)
- **Dynamic:**
  - Time: (“Data is Beautiful,” 2019)
    - 1980s (Fortran, Pascal, Ada); 1990s (C, C++); 2000s (Java, PHP); 2020s (Python)
  - Field: (Berkely Extension, 2023)
    - E-commerce (Java); OS (Rust); SysAdmins (Perl); Data science (Python), ...

➡ Beginners constantly question their choices → change language frequently without being productive

Challenges for Beginners

Learning a Language is Difficult!

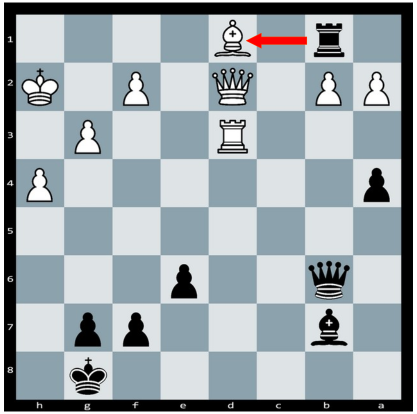
- **Excessive amount time necessary to grasp syntax**—here referring to C and VisualBasic.NET (Al-Imamy et al., 2006)
  - The C-style syntax has influenced many languages (e.g. Java, PHP, Go or Swift), but is **challenging for beginners** (Denny et al., 2011; Stefik & Siebert, 2013)

- Programming Classes: **High dropout rates** and **poor outcomes**
  - Students cannot create loops after several semesters (Robins et al., 2003)
  - Become disillusioned with programming (Garner, 2002)

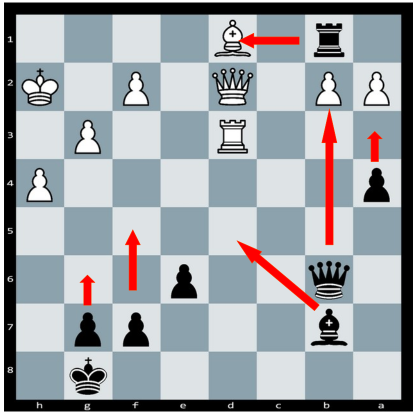
➡ What makes learning a programming language so difficult?

Human Expertise and Problem-solving Skills

- Human expertise and problem-solving skills, are based on knowledge stored as so-called **schemata** in our **long-term memory** (Sweller & Van Merriënboer, 2005; Garner, 2002)
  - **Schemata**: Any exiting knowledge that can be treated as a single element or piece of information—e.g. word, pattern, formula, or concept... (Garner, 2002)



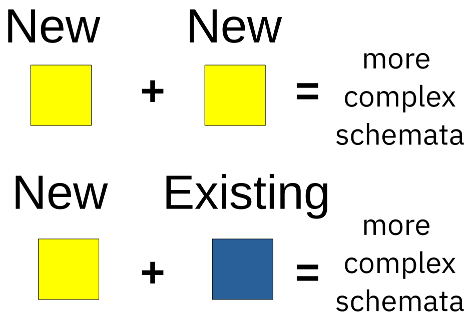
Expert: Recognizes pattern and “automatically” retrieves solution



Beginner: Needs to “actively” consider different moves/element

Learning and Teaching

- **Learning:** Disconnected pieces/elements of information are bundled/chunked into a **more complex schemata** (Paas et al., 2003)
  - Requires active thinking → **free working memory capacity** (Sweller & Van Merriënboer, 2005)
  - Schemata can be treated as a single element in working memory



New 🟨 + New 🟨 = more complex schemata

New 🟨 + Existing 🟦 = more complex schemata

- **Goal of teaching:** (Sweller & Van Merriënboer, 2005)
  - Enable the construction of more complex schemata
  - Facilitate their automation through practice

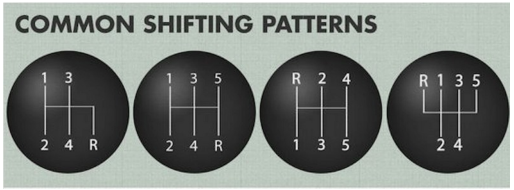
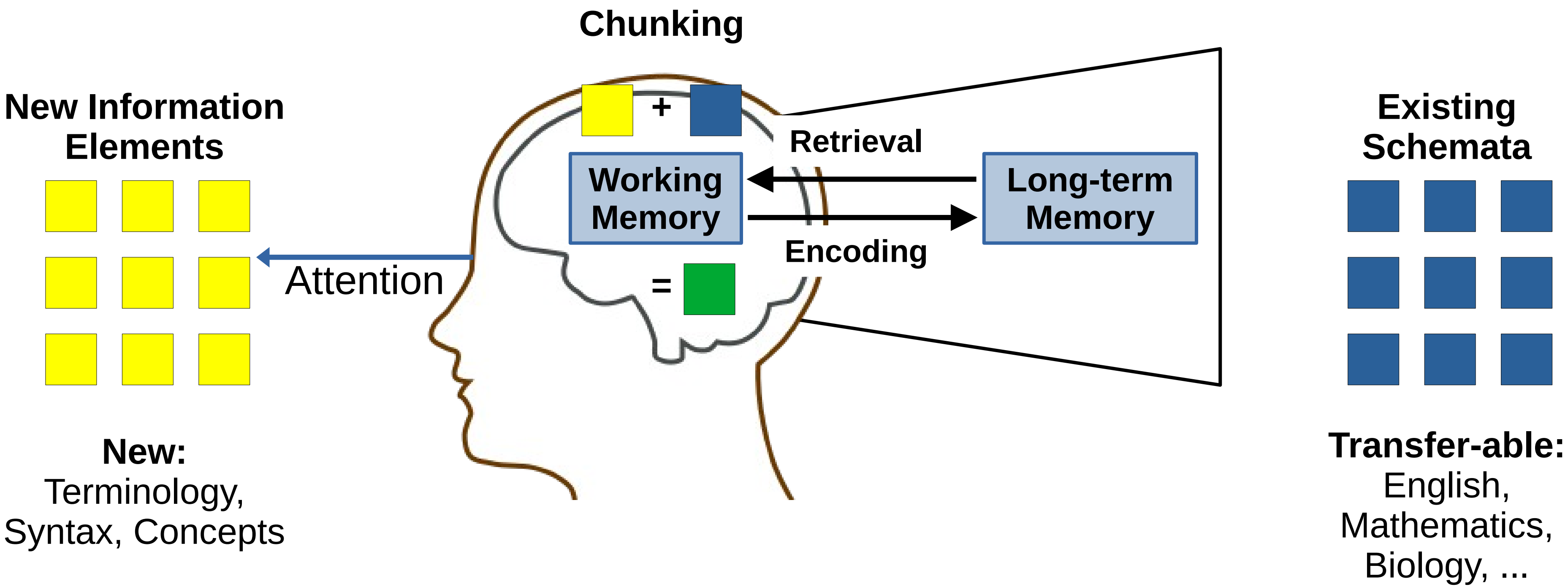


Image Source: *https://149637512.v2.pressablecdn.com/wp-content/uploads/Screen-Shot-2017-11-15-at-7.00.05-PM.png*

Working Memory as a Bottle Neck, 1



New Information Elements

Attention

Chunking

+

=

Working Memory

Retrieval

Encoding

Long-term Memory

Existing Schemata

New:
Terminology,
Syntax, Concepts

Transfer-able:
English,
Mathematics,
Biology, ...

Working Memory can only handle **7 (± 2)** elements (schemata) for a duration between **10-15 seconds**

Working Memory as a Bottle Neck, 2

- Examples:
  - **Learning Digits:**
    - 2 – 0 – 0 – 3 – 2 – 0 – 0 – 4 → **individually is hard!**
    - 200 – 3 – 200 – 4 → **chunking is easier!**
    - 20 – 03 – 2004 → **with knowledge is easiest! (25th anniversary)**
  - **Learning a Natural Language:**
    - Hello—beloved—world—my → **learning vocabulary (chunks) is fairly easy!**
    - Hello, my beloved wold! → **learning grammar / equivalent to syntax is hard!**
      - <u>especially if:</u> words and their semantic is new (no previous knowledge)
      - <u>especially since:</u> interaction between needs to be considered (adds new elements)

➡ Building on **prior knowledge**, **reducing the number of elements** and **their interactivity**.

Build on Previous Knowledge – Some Examples, 1

- Literal aspects rooted in English or Math are perceived as easier (Stefik & Siebert, 2013):
  - `REPEAT` or `LOOP` is easier than `FOR`
  - Single equal sign (“=”) is easier than double equal sign (“==”)
- Abbreviations
  - Python (“forced cleverness”):
    - *`.strip(), .lstrip()`*
      - Beginner has to learn what “`l`” stands for
      - new: “`l`” + known: “`strip`” + context
  - ooRexx:
    - `~strip(), ~strip(“leading”), ~strip(“l”)`
      - Beginner can use “leading” and can later switch to “l”
      - 2 x known: “strip” + “leading” + context

Build on Previous Knowledge – Some Examples, 2

Python

```
Var = 1
if Var == 1:
    print("Yes")
else:
    print("No")
```

- Need for different equal signs
  - <u>otherwise:</u> Syntax errors
- **Indentions have semantic meaning**
  - <u>clever</u>: reduce elements (do, end, …)
  - <u>but</u>: exiting knowledge is not applicable!

ooRexx

```
Var = 1
if Var = 1 then say     "Yes"
           else say     "No"
```

- Single equal sign can be used
  - as known from Math
- **Free-form Characteristic**
  - as natural language
  - exiting knowledge is applicable!

Build on Previous Knowledge – Some Examples, 3

Python
(Case Dependence)

```
Oranges = 1
print(Oranges)
print(oranges) #NameError: name
'oranges' is not defined. Did you mean:
'Oranges'?
```

- **Example:** Variables
  - **Oranges** and **oranges** are here two different "things"
  - Existing knowledge from natural language is not applicable!

ooRexx
(Case Insensitivity)

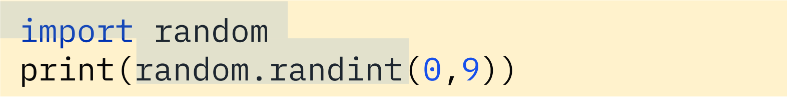
```
Oranges = 1
say Oranges
SaY oranges
```

- **Example:** Variables
  - **Oranges** and **oranges** are the same "thing"
  - Existing knowledge from natural language is applicable!
- Applies to any aspect of the language

Reduce Interactivity – Some Examples, 1

- Build-in functions and information search

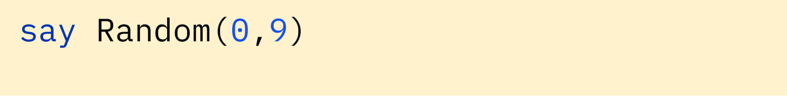
Python



```
import random
print(random.randint(0,9))
```

- Beginners need to consider more interacting elements
  - e.g. `import` must happen at the beginning, otherwise:
    - # NameError: name 'random' is not defined
  - Errors/messages should be understandable (McIver & Conway, 1996)

ooRexx



```
say Random(0,9)
```

- Beginners need to consider less interacting elements
- All knowledge in a single reference manual → reduces cognitive load by minimizing search (Sands, 2019)
  - Good Manual: syntax diagram, description, working example

Roundup

- Working memory capacity is very limited → ooRexx language characteristics reduce cognitive burden/facilitate learning
  - Free-form and case-insensitive characteristics
  - ooRexx makes existing knowledge applicable (e.g. Math, literal English)
  - Powerful build-in functions all in one manual
  - Understandable error messages
- Be consistent with abbreviations and also allow long derivatives
  - Example: String Class Methods
    - `~changeStr` not possible: `~changeString`
    - `~makeString` not possible: `~makeStr`
- Thanks for Listening: <till.winkler@wu.ac.at>

References

- List of programming languages. (2023, May 27). In Wikipedia. https://en.wikipedia.org/wiki/List_of_programming_languages.
- PYPL. (2023). PYPL PopularitY of Programming Language. Retrieved from https://pypl.github.io/PYPL.html.
- Data is Beautiful. (2019, October 7). Most Popular Programming Languages 1965 – 2019 [Video file]. Youtube. https://www.youtube.com/watch?v=Og847HVwRSI
- Berkely Extension. (2023). 11 Most In-Demand Programming Languages. Retrieved from https://bootcamp.berkeley.edu/blog/most-in-demand-programming-languages/
- Al-Imamy, S., Alizadeh, J., & Nour, M. A. (2006). On the development of a programming teaching tool: The effect of teaching by templates on the learning process. Journal of Information Technology Education: Research, 5(1), 271-283.
- Denny, P., Luxton-Reilly, A., Tempero, E., & Hendrickx, J. (2011, June). Understanding the syntax barrier for novices. In Proceedings of the 16th annual joint conference on Innovation and technology in computer science education (pp. 208-212).
- Stefik, A., & Siebert, S. (2013). An empirical investigation into programming language syntax. ACM Transactions on Computing Education (TOCE), 13(4), 1-40.
- Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. Computer science education, 13(2), 137-172.
- Garner, S. (2002). Reducing the cognitive load on novice programmers (pp. 578-583). Association for the Advancement of Computing in Education (AACE).
- Sweller, J., & Van Merriënboer, J. J. G. (2005). Cognitive load theory and complex learning: Recent developments and future directions. Educational Psychology Review, 53(3), 147-177.
- Paas, F., Tuovinen, J. E., Tabbers, H., & Van Gerven, P. W. (2003). Cognitive load measurement as a means to advance cognitive load theory. Educational psychologist, 38(1), 63-71.
- McIver, L., & Conway, D. (1996, January). Seven deadly sins of introductory programming language design. In Proceedings 1996 International Conference Software Engineering: Education and Practice (pp. 309-316). IEEE.