Teaching and Learning Python by Comparative Visualization

Gulzira Izbassova, Syeda Umaima Minhaj, and Eugene Pinsky(✉)

Department of Computer Science, Boston University, Metropolitan College 1010 Commonwealth Avenue, Boston, MA 02215, USA
{gulzira,sminhaj,epinsky}@bu.edu

Abstract. Python is built with very few data types and constructs. Drawing on an earlier approach to learning spoken language through picture comparisons, we propose to teach and learn Python similarly, By using simple diagrams to represent data types and constructs, we associate each new Python concept with a pair of simple diagrams. Teaching and learning Python is accomplished incrementally by logically connecting each new Python concept to differences in the visualizations. We believe that with examples and corresponding visual comparisons, our approach is effective for Python learners of all levels, helping them gain a better understanding and mastery of the language.

Keywords: Python · visualization · learning

1 Background

In the early 1970s, a Russian-born physicist, Rudolf Tenenbaun, published a series of books [15] on Learning Spoken English with pictures. In his method, a student will memorize words and sentences (tape-recorded by the teacher) where each word or sentence is described by a picture. Each picture consists of two parts and its meaning is inferred ("computed") by the comparison of its right and left half. As a simple example, consider one such picture in Fig. 1: This picture says, "The box is big" - this is conveyed by the comparison between the right and the left parts of this picture. The pictures can become more complex, but each part can be computed by the proper comparison. Consider the picture in Fig. 2: This picture says, "The box is heavy". It consists of two halves, where each half consists of two "basic pictures". In the left half, we compute "the person can lift the box", whereas in the right half, "the person cannot lift the box". The meaning of this, more complex, picture is "The Box is heavy".

From these more elementary pictures, students learn more advanced pictures and sequences of pictures. A typical example is shown in Fig. 3.

The authors would like to thank Boston University Metropolitan College for their support.

T. Zlateva and G. Tuparov (Eds.): CSECS 2024, LNICST 609, pp. 310–331, 2025.
https://doi.org/10.1007/978-3-031-84312-9_21

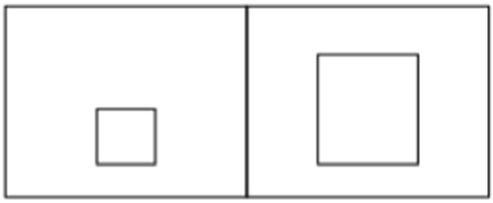


Fig. 1. Illustration to "The box is big"

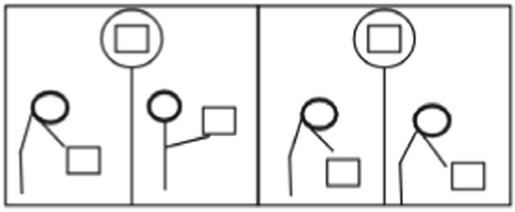


Fig. 2. Illustration to "The box is heavy"

Remembering sentences for these elementary pictures, students can construct a series of sentences. Since pictures are "computed" by comparison and associated with pre-recorded words and sentences, a student can substitute appropriate pictures or sentences and will be in a position to start speaking through these picture associations.

Tenenbaum's method is based on a few basic principles

1. simple diagrams, employing relatively few words in examples (e.g., woman, man, girl, boy, box, ball, etc.)
2. meaning is done by comparison
3. diagrams are simple - no color, very few primitives
4. each picture conveys a single idea

The last item is very important. Learning one concept at a time, with its own picture, is believed to be more effective than overwhelming students with fancy color diagrams presenting many ideas at once.

2 From English to Python

We believe that the same principles should be used in learning Python. There are a lot of similarities with the Tenenbaum's method.

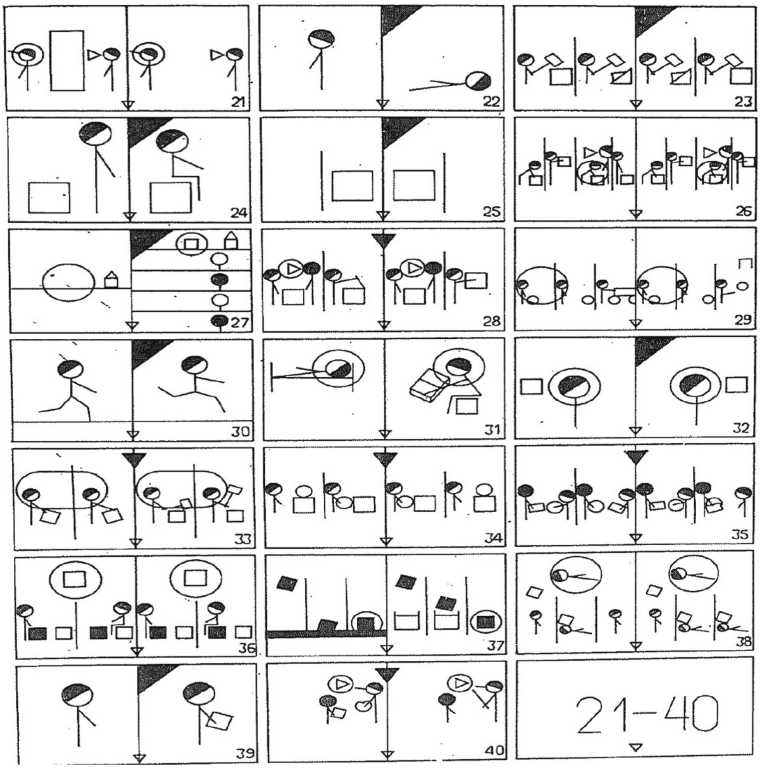


Fig. 3. Learning Stories and Passages (from Tenenbaum's book)

1. Python has very few data types
2. major concepts could and should be learned one at a time
3. These concepts are better learned through simple diagrams
4. diagrams are best understood by comparison

We have started such an effort. In this innovative project, we embark on a visually captivating journey to demystify Python's data types by representing them as distinct shapes. We aim to enhance the accessibility and ease of learning Python by relying on comparative visualization that aids comprehension.

At the core of our approach are simple diagrams and comparisons. Through meticulous coding in pure Python, we have crafted symbolic shapes for various data types, fostering a seamless connection between code and visualization:

1. **Numeric Types (Integer, Float, Complex):** Symbolized by a "shining" diamond, these fundamental numeric data types shine as the building blocks of mathematical operations.
2. **Character Type:** Encircled gracefully, the character type takes the form of a circle, signifying its singular and concise nature.

3. **Boolean Type:** Represented by a dynamic arrow oscillating between up and down, visualizing the Boolean's binary nature, alternating between true and false states.
4. **List:** Two rectangles interlinked by a line portray the versatility of lists, highlighting their ability to store and manage sequential data.
5. **Tuple:** Two trapezoids connected with a line embody tuples' ordered yet immutable nature, showcasing their significance in Python data structures.
6. **Set:** A sturdy triangle stands tall, symbolizing the unordered and unique elements that define sets.
7. **String:** Two interconnected circles, united by a line, elegantly capture the essence of strings, demonstrating their ability to concatenate and form cohesive textual entities.
8. **Dictionary:** An oval shape with lines extending to various keys represents the dynamic nature of dictionaries, emphasizing their role in efficient key-value pairing.

We believe that learning Python using comparative visualization with the above representations gives us powerful educational tools catering to diverse learning needs. Studying these diagrams will provide valuable insights into Python, enhancing our understanding and proficiency in the language. In the rest of the paper, we present our diagrams, and exercises and compare our approach with other visual learning of Python.

3 Basic Diagrams

There are a total of 10 data types in Python. These are divided into primitive data types and collections.

3.1 Primitive Data Types

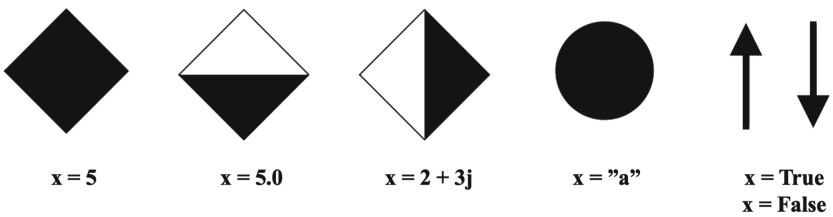


Fig. 4. Diagrams for the primitive data types

There are five primitive data types, as shown in Fig. 4.

1. Integer

```
x = 5
```

2. Float

```
x = 5.0
```

3. Complex

```
x = 2 + 3j
```

4. Character

```
x = "a"
```

5. Boolean

```
x = True
x = False
```

3.2 Collections

There are 5 collections in Python: list, string, set, tuple, and dictionary In addition, there are two special objects: None and range. These collections and their diagrams are described below

1. list data type:

```
x = [1, 2]
```

Lists are represented by two rectangular shapes connected by a straight line, as shown in Fig. 5.

Fig. 5. Diagram for a simple list with two integers

Here is an example of a typical exercise for students
- Create and visualize an empty list.
- Create and visualize a list with five integers.

– Create and visualize a list with five strings.

2. String Data Type

```
x = "Hello World"
```

Strings are represented by two circular shapes connected by a straight line, as shown in Fig. 6.

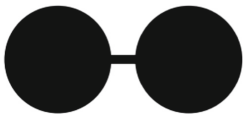


Fig. 6. Diagram for a string

3. Set Data Type

```
x = {1}
```

Set data types are represented by a triangle. If a set contains any data, it is represented by the particular shape within the triangle. This is illustrated in Fig. 7.

Fig. 7. A diagram of a set containing a single number

Here is an example of exercise/quiz
– Create and visualize a set with five integers.
– Create and visualize a set with five strings.
– Create and visualize a set with a mix of data types.

4. Tuple Data Type

```
x = (1, 2)
x = x + (3,)
```

Fig. 8. A 2-element vs. 3-element tuple with integers

Here, we have two trapezoid shapes connected in a straight line. This is the representation of a Tuple, as shown in Fig. 8.

The picture on the left in Fig. 8 depicts x = (1,2), and the picture on the right depicts x = x + (3,). If you notice the connecting lines in both pictures, one is on the top, and the other is at the bottom. We are representing "before" and "after" images of x, where after adding a value to the tuple, the connecting lines are now at the bottom.

Here is an example of an exercise/quiz
- Create and visualize a tuple with three strings.
- Create and visualize a tuple with a mix of data types.
- Create and visualize a tuple of tuples (nested tuples).

5. Dictionary Data Type

```
x = {"B": "ASDFGHJ", 4 :[1]}
```

Dictionaries are represented by the data types associated with respect to the keys and values connected by a straight line with lines at the bottom pointing the keys to its values. This is shown in Fig. 9.

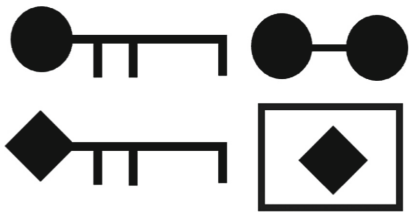


Fig. 9. Diagram of a dictionary with a string and a list as values

List, Set, and Dictionary are Mutable data types, whereas String and Tuple are Immutable data types.

3.3 Learning Data Types by Comparative Visualizations

Once the students are introduced to the datatypes, we can design examples to illustrate similarities and differences. Each example should present a simple concept, and this is illustrated by a comparative visualization.

1. Concatenate a string. Since strings are immutable, we can add elements to a string by using concatenation.

```
x = "abc"
x = "abc" + "d"
```

This is illustrated in Fig. 10

Fig. 10. Illustration of string concatenation

In the above example, we can see strings represented by two circles connected by a straight line. Now, observe the image on the right where an element is added to the string, and the connecting lines are shifted to the bottom. This is an example of an immutable data type.

2. append to a list

```
x = [1, 2]
x.append(3)
```

This is illustrated in Fig. 11.

Fig. 11. Diagram on appending to a list

This example shows that it is possible to grow the list by adding an element

3. concatenate two lists

```
x = [1,2]
x = x + [3]
```

This is illustrated in Fig. 12.

In the first example, notice the connecting lines on the pictures are on the same level, i.e., this is mutable as we use “append.” However, in the second

example, notice the connecting lines on the after picture is at the bottom, i.e. this is immutable as we add elements to the list using x = x + [3]. This is summarized in Fig. 13.

Fig. 12. Diagram on concatenating lists

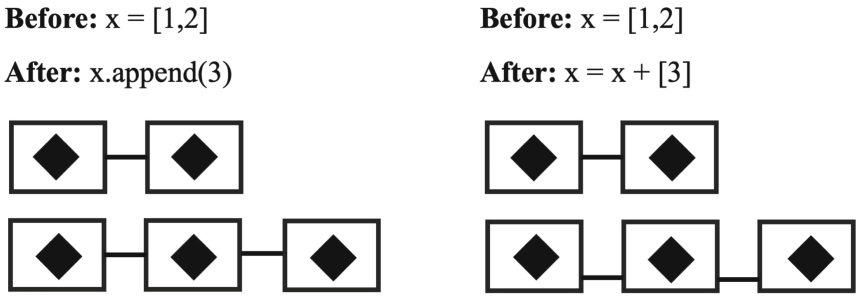


Fig. 13. Append vs. Concatenate in lists

4. remove an element from a list

```
x = [1, "A", 3]
x.pop(3)
```

This is illustrated in Fig. 14.

Fig. 14. Removing an element from a list

In the above example, the picture on the left and right represent a before and after representation of popping an element from a list.

5. Casting an “integer” data type to a “string” data type

```
x = 3
x = str(3)
```

This is illustrated in Fig. 15.

6. Casting a "list" to a "tuple" data type

Fig. 15. Casting integer to a string

```
x = [1, 2.5, 3]
x = tuple(x)
```

This is illustrated in Fig. 16.

Fig. 16. Casting a list to a tuple

Observe that in the above example the shape of the list which are rectangles connected with the help of a straight line are now changed to trapeziums (representing a tuple) connected with straight lines.

Here are some exercises/quizzes
- Access and visualize the last element of a list.
- Access and visualize the third element of a list.
- Access and visualize a range of elements in a list (slicing).

7. Concatenating two tuples Concatenate a Tuple. Since tuples are immutable data types, we can add elements to a tuple by concatenation.

```
x = (1, 2)
x = (1, 2, 3)
```

Fig. 17. Concatenating two tuples

This is illustrated in Fig. 17.

Here are some examples of exercises/quizzes
- Access and visualize the last element of a tuple.
- Access and visualize the third element of a tuple.
- Access and visualize a range of elements in a tuple (slicing).

8. create a list with different data types

```
x_list=[True, 4.5, "A"]
y_list=[[True, False],{1,2},["apple"]]
```

This is illustrated in Fig. 18.

Fig. 18. A list primitive types vs. list with collections

9. create lists with strings and characters

```
char_list = ["h", "e", "l"]
str_list  = ["hi", "hello", "hola"]
```

This is illustrated in Fig. 19.

Fig. 19. List of characters vs. list of strings

10. create a list of numbers

```
int_list   = [1, 2, 3]
float_list = [1*3.5, 2*3.5, 3*3.5]
```

This is illustrated in Fig. 20.

Here are some examples of exercises/quizzes
- Start with a list, e.g., [True, 3.14, “B”, 2], and then append an element to it (e.g., “abc”). Visualize the list before and after the append operation.
- Create and visualize a list of squares of numbers from 1 to 10 using a list comprehension.
- Create and visualize a list of even numbers from 1 to 20 using a list comprehension.

Fig. 20. List of integers vs. list of floats

4 “If” Statement

The “**if**” statement is a fundamental control structure in Python that allows for conditional execution of code blocks. It evaluates a condition (an expression that returns a Boolean value), and if the condition is true, it executes a block of code. The “**if**” statement can be extended with “**elif**“ (short for “**else if**”) and “**else**” blocks to handle multiple conditions and provide a default action.

4.1 Basic If Statement and If-Else Statements

if statement: Check and visualize if x is greater than 5. If it is, print ”x is greater than 5”.

```
x = 10
if x > 5:
    print("x is greater than 5")
```

if-else statement: Check and visualize if x is greater than 5. If it is, print ”x is greater than 5”, else print ”x is less than 5”.

```
x = 3
if x > 5:
    print("x is greater than 5")
else:
    print("x is less than 5")
```

Both constructs are illustrated in Fig. 21

In the first example, the Python statement "x" is represented by an integer and if it is greater than another integer then a string is printed.

In the second example, the Python statement "x" is represented by an integer and if it is greater than another integer then a string is printed, else if that is not true then print another string.

Here, "?" is the representation of an if statement.

4.2 Elif Statement

Problem: Categorize a number as "Positive", "Negative", or "Zero" and visualize it.

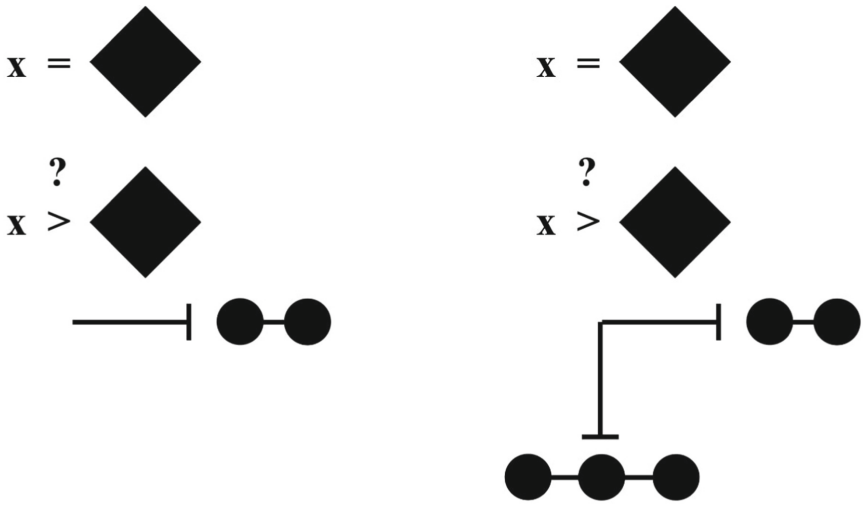


Fig. 21. "if" vs. "if-else" diagram

```
x = -1
if x > 0:
    print("Positive")
elif x < 0:
    print("Negative")
else:
    print("Zero")
```

In the following example, the Python statement "x" is represented by an integer, and if it is greater than another integer, then a string is printed; else, if the value of x is less than another integer, then print the given string. Otherwise, if both the conditions are not true, then print another string.

The question mark "?" is the representation of a comparison statement. This is illustrated in Fig. 22.

5 Loops

Loops in Python are used to execute a block of code repeatedly, either for a fixed number of times or until a certain condition is met. Python provides two primary types of loops: the "for" loop and the "while" loop.

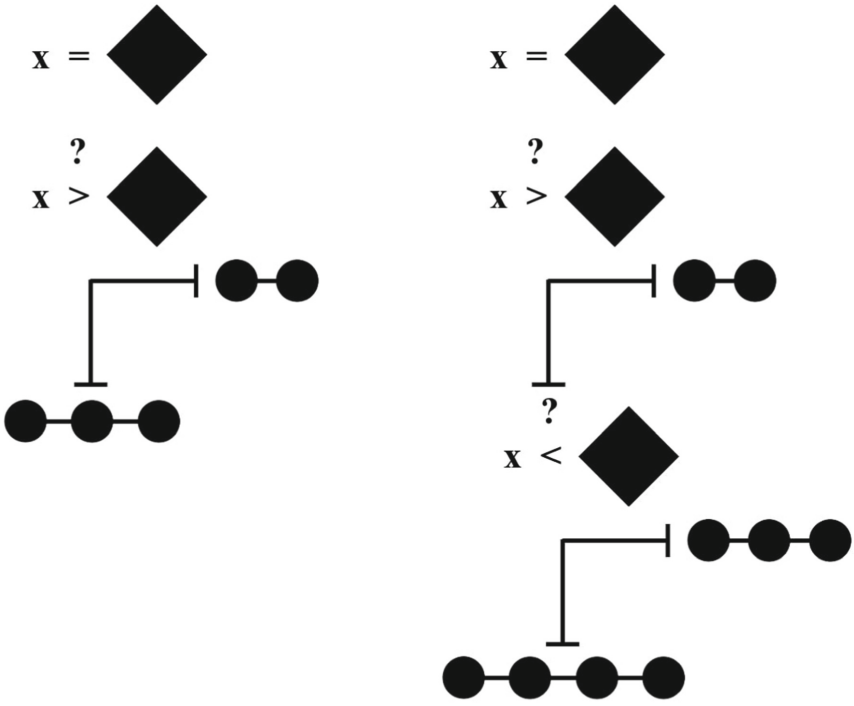


Fig. 22. "if-else" vs. "if-elif-else"

5.1 The For Loop

The "for" loop in Python is used to iterate over a sequence (such as a list, tuple, dictionary, set, or string) or other iterable objects. Iterating over a sequence means going through each item in the sequence one by one. Consider the following two problems:

– Problem 1: Print each character in a string and visualize it.

```
#for loop
for x in "Hello":
    print(x)
```

– Problem 2: Print numbers 1 through 5 and visualize it.

```
#for loop
for x in range(1, 6):
    print(x)
```

Both examples are illustrated in Fig. 23.

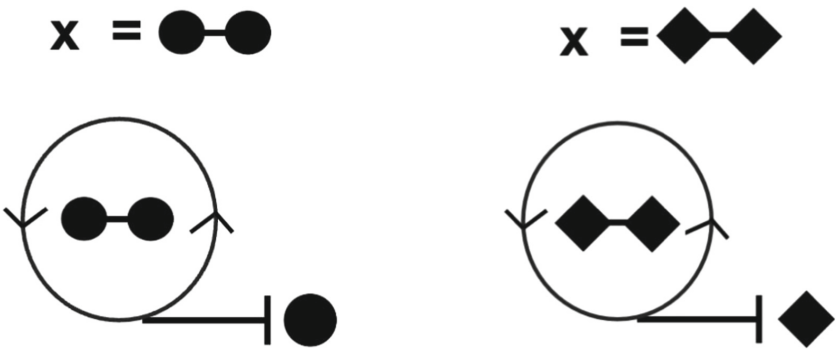


Fig. 23. Iterating over strings vs. iteration over ranges with "for"

5.2 While Loop

The "while" loop in Python is used to execute a block of code as long as a condition is true. It's useful when you want to continue looping until a particular state is reached or until some condition becomes false.

– Problem 1: Print each character in a string

```
#while loop
x = "Hello"
while x:
    print(x[0])
    x = x[1:]
```

– Problem 2: Print numbers 1 through 5

```
#while loop
x = 1
while x < 6:
    print(x)
    x += 1
```

Both problems are illustrated in Fig. 24.

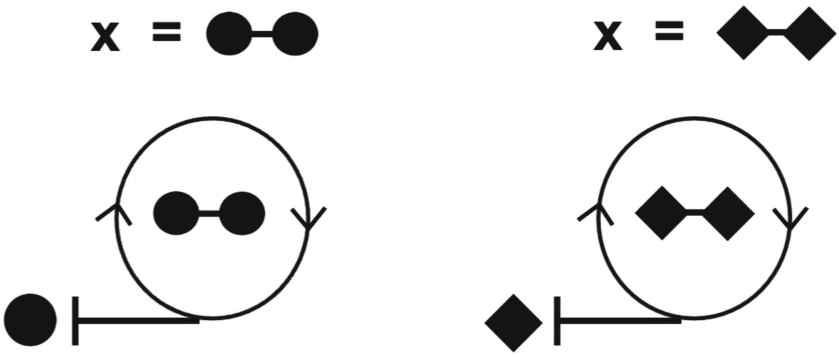


Fig. 24. Iterating over strings vs. iteration over ranges with "while"

5.3 For vs. While Loop with String

Objective: Demonstrate and visualize iterating over a sequence with a "for" loop and a "while" loop.

– Problem: Print each character in a string and visualize it.

```
# for loop
for x in "Hello":
    print(x)

# while loop
x = "Hello"
while x:
```

```
print(x[0])
x = x[1:]
```

In the following diagram in Fig. 25, the "for" loop is represented on the left and the "while" Loop on the right.

Notice the arrows of the circles in both examples on the left and right and following opposite directions. The For Loop on the left has a circle in a counter-clockwise direction loop and the While Loop on the right has a circle in a clock-wise direction loop.

Finally, the straight line from the outside of each circle represents the final print statement after following the loop.

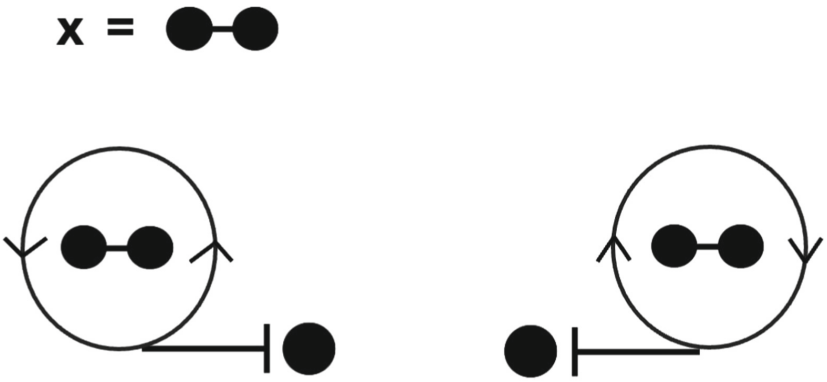
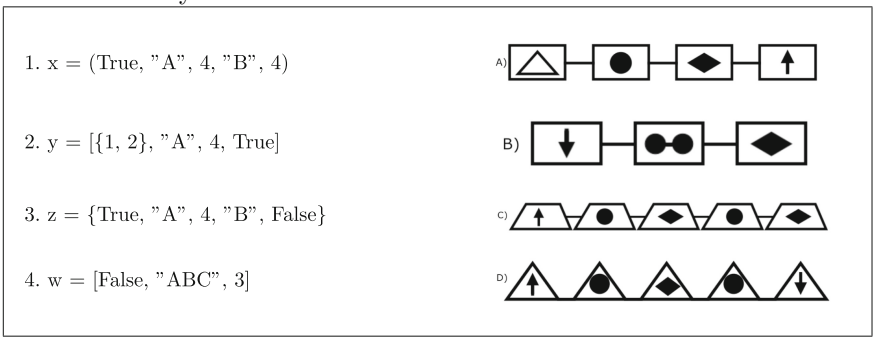


Fig. 25. "for" vs. "while" loop

6 Examples of Exercises to Matching Code and Diagrams

In this section, we present a few examples of learning exercises in our approach.

1. Each shape represents a piece of code. Your goal is to match them correctly based on what you've learned so far.



2. Choose the correct image to be added to the diagram below with the help of the code snippet:

v = [[1,1] , ”ABC”, ’a’]

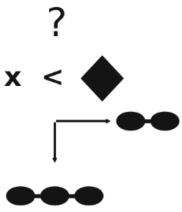
A.

B.

C.

D.

3. Choose the correct code snippet that matches the diagram below.

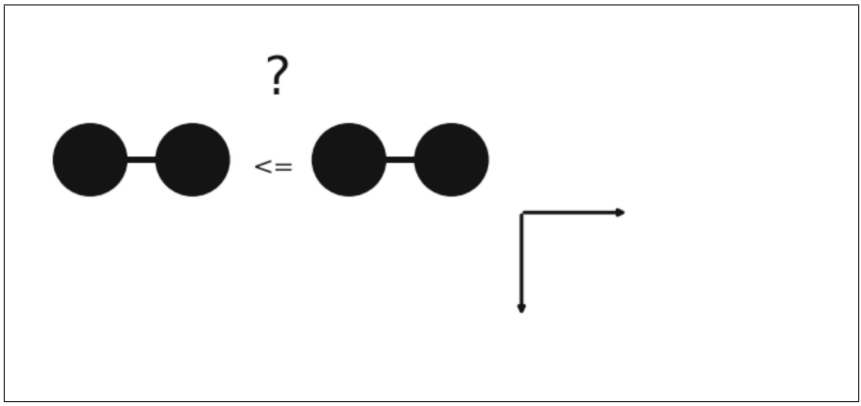


A. x = 6 if x ¡ 3: print(”x is less than 3”) else: print(”x is greater than 3”)
B. x = 6 if x ¡ 3j+1: print(”x is less than 3”) else: print(”x is greater than 3”)
C. x = 6 if x ¡ ”abc”: print(”x is less than 3”) else: print(”x is greater than 3”)
D. x = 6 if x ¡ 3.0: print(”x is less than 3”) else: print(”x is greater than 3”)

4. According to the code snippet below, complete the following diagram for missing shapes.

```
string1 = "hello"
string2 = "world"\xm{\para}

if string1 <= string2:
    print("Perform task 1")
else:
    print("Perform task 2")
```



7 How Is Our Approach Different?

We believe that learning Python by comparative visualization offers distinctive advantages:

1. **Research on Visual Learning:** Numerous studies have demonstrated the effectiveness of visual learning in education. For example, studies in [2,2] found that visual explanations can significantly enhance learning outcomes by improving comprehension and retention of information.
2. **Memory Retention with Visual Aids:** Many research studies ([9,10] suggest that visual aids, particularly those that are distinctive or unusual, can improve memory retention. This could apply to comparative visualizations in learning Python, as it may make the concepts more memorable for learners.
3. **Engagement and Comprehension:** Many studies (e.g. [16]) have shown that organized visual displays can improve comprehension and engagement among learners. Comparative visualizations could enhance engagement by presenting Python concepts in a visually appealing and organized manner, aiding in comprehension.
4. **Application of Cognitive Load Theory:** Comparative visualizations can also align with principles of cognitive load theory, which suggest that presenting information in a visually organized and comparative manner can reduce cognitive load and enhance learning outcomes. While there may not be specific studies on Python learning using comparative visualization, research in cognitive load theory supports the potential effectiveness of this approach.
5. **Enhanced Understanding:** Comparative visualizations allow learners to compare different Python concepts side by side, facilitating a deeper understanding of how they relate to each other. For example, comparing different data types or control flow structures visually can help learners grasp their similarities and differences more effectively.

6. **Clarity and Conciseness:** Visual representations can often convey complex ideas more clearly and concisely than textual descriptions alone. By comparative visualizations, this approach may streamline the learning process by presenting information in a more digestible format, reducing cognitive load, and enhancing comprehension.
7. **Memory Retention:** Visual learning has been shown to improve memory retention (e.g. [16]). By presenting Python concepts through comparative visualizations, our approach may help learners retain information more effectively by providing memorable visual anchors for abstract concepts.
8. **Engagement:** Visual content can enhance learner engagement by making the learning experience more interactive and stimulating. Comparative visualizations may encourage learners to actively explore and analyze Python concepts, leading to a more immersive learning experience.
9. **Application-Oriented Learning:** Comparative visualizations can facilitate a more application-oriented approach to learning Python. By relating different Python concepts to real-world scenarios, the approach may help learners develop practical problem-solving skills and apply their knowledge more effectively in programming projects.

8 Simplicity of Our Approach

Our approach relies on very simple basis diagrams and corresponding comparisons. Let us contrast our approach with typical visual Python books [1,3–6,8,12] to name just a few. Consider one such book, "Learn Python Visually" by Ivelin Demirov [4]. This book, like many others, presents many visualization techniques to explain various Python concepts. However, in these books, the examples are too complex, have too many details, and try to present multiple concepts at the same time.

We illustrate this with two examples. For each example, we present a visualization from this book on the left and our visualization on the right.

– **Example 1:** "list" construct shown in Fig. 26.

Fig. 26. Complex vs. simple representation of the "list" construct

– **Example 2:** "while loop" construct shown in Fig. 27.

The diagrams on the left in Figs. 26 and 27 uses many different colors, such as green, red, and yellow. Many varied shapes, are employed to illustrate a single list. In addition, multiple concepts (lists are mutable, strings are immutable)

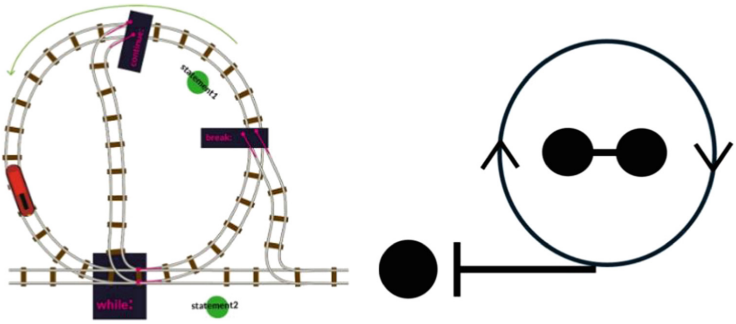


Fig. 27. Complex vs. simple representation of the "while" construct

are presented at once. Contrast this with our presentation of the list or while constructs via the comparative visualization approach. We believe that our approach is much simpler and much easier with the help of just a few basic shapes and diagrams. We believe that using many shapes, colors, and complex figures lead to confusion and hinders comprehension.

We can back our arguments with a number of studies. Research in cognitive psychology (e.g. [14]), suggests that presenting information in a simpler format reduces cognitive load, allowing learners to focus more effectively on understanding the material. When visualizations become too complex, they may overload working memory, leading to decreased comprehension. Numerous studies have shown that visual learning aids when appropriately designed, can enhance learning outcomes. However, these studies often emphasize the importance of clarity, simplicity, and relevance in visual materials. Visual clutter or unnecessary complexity can impede learning rather than facilitate it.

By contrast, our approach to teaching and learning Python adopts a simpler approach: We utilize only black and white colors and we employ a single shape for each data type. Moreover, we provide comparative visual explanations for each concept, as depicted in the accompanying picture. This straightforward method facilitates easy learning and understanding:

1. **Comparative Learning:** Comparative learning, where learners are presented with side-by-side comparisons of concepts, has been shown to promote deeper understanding and retention [7,9]. By highlighting similarities and differences between concepts, learners can build more robust mental models and make connections more easily.
2. **Usability Principles:** In the design field, principles such as simplicity, consistency, and clarity are emphasized to improve user experience and comprehension [11,13,14]. These principles can be applied to educational materials, including visualizations, to ensure they are effective and user-friendly.

9 Conclusion

We strongly believe that the proposed method of Learning Python by Comparative Visualization offers the advantage of presenting Python concepts in a visually engaging and comparative manner, fostering deeper understanding, improved retention, and practical application of programming concepts.

References

1. Barry, P.: Head First Python: A Learner's Guide to the Fundamentals of Python Programming, A Brain-Friendly Guide, 3rd Edition. O'Reilly Media, September (2023)
2. Bobek, E., Tversky, B.: Creating visual explanations improves learning. Cogn. Res.: Principles Implic. **1**(1), 1–14 (2016). https://doi.org/10.1186/s41235-016-0031-6
3. Briggs, J.R.: Python for Kids, 2nd Edition: A Playful Introduction to Programming. No Starch Press (Nov 2022)
4. Demirov, I.: Learn Python Visually. CreateSpace Independent Publishing Platform, United States (2015)
5. Fehily, C.: Python: Visual QuickStart Guide . Peachpit Press (Oct 2001)
6. Yang, H.: Kids Learning Python: Kids Learn Coding Like Playing Games. Independently Published, N.p. (2020)
7. Ligon, F., Tannenbaum, E., Rodgers, C.: More Picture Stories: Language and Problem-Posing Activities for Beginners. Longman, South Korea (1992)
8. Matthes, E.: Python Crash Course, 3rd Edition: A Hands-On, Project-Based Introduction to Programming. No Starch Press (Jan 2023)
9. McDaniel, M.A., Einstein, G.O.: Bizarre imagery as an effective memory aid: The importance of distinctiveness. J. Exp. Psychol. Learn. Mem. Cogn. **12**(1), 54–65 (1986)
10. Patton, W.W.: Opening students' eyes: Visual learning theory in the socratic classroom. Law Psychol. Rev. **15**, 1–18 (1991)
11. Perfors, A., Tenenbaum, J.B., Regier, T.: The learnability of abstract syntactic principles. Cognition **118**(3), 306–338 (2011)
12. Mardiyyah Rufai, A.: Creating patterns in python -learnpythonthroughprojects: Series 7, (2020)
13. Daniel, L.: Schacter. Searching for Memory. Basic Books, New York (1966)
14. Sweller, J., Ayres, P., Kalyuga, S.: Cognitive Load Theory. Springer New York, New York, NY (2011). https://doi.org/10.1007/978-1-4419-8126-4
15. Tenenbaum, R.: Speaking and Spelling Through Picture Problems. Physical-Technical Institute, St. Petersburg, Russia (1970)
16. Verdi, M.P., Johnson, J.T., Stock, W.A., Kulhavy, R.W., Whitman-Ahern, P.: Organized spatial displays and texts: effects of presentation order and display type on learning outcomes. J. Experiment. Educ. **65**, 303–317 (1997)