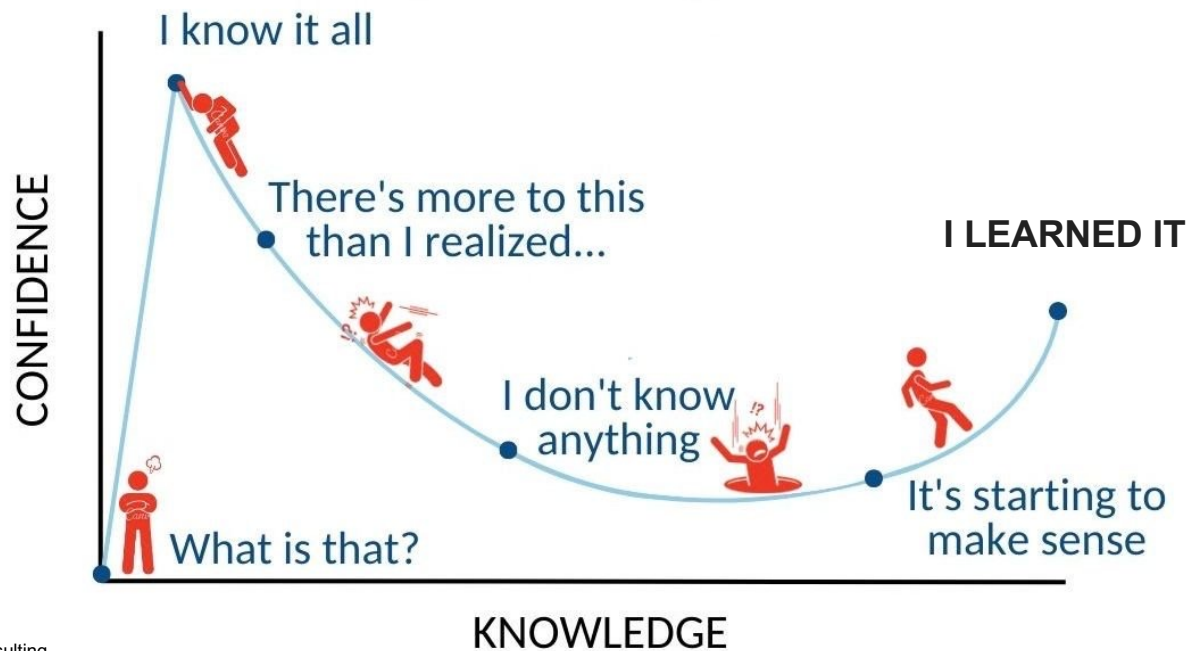


correlation.:one
TECH FOR JOBS

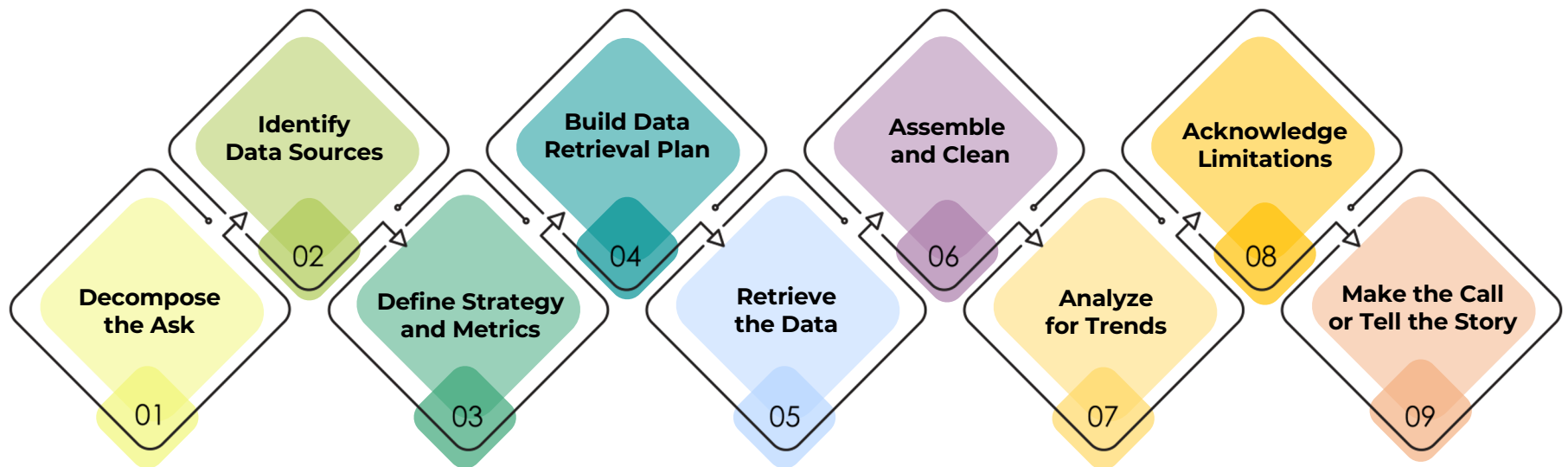
Dunning-Kruger Effect



Credit: BVA Nudge Consulting

Analytics Paradigm

Regardless of type or industry, this paradigm provides a repeatable pathway for effective data problem solving.



DuckDB



Documentation ▾ Resources ▾ GitHub ★ 25.1k

Support



DuckDB is a fast in-process database system

Query and transform your data anywhere
using DuckDB's feature-rich SQL dialect

Installation ▾

Documentation

SQL Python R Java Node.js

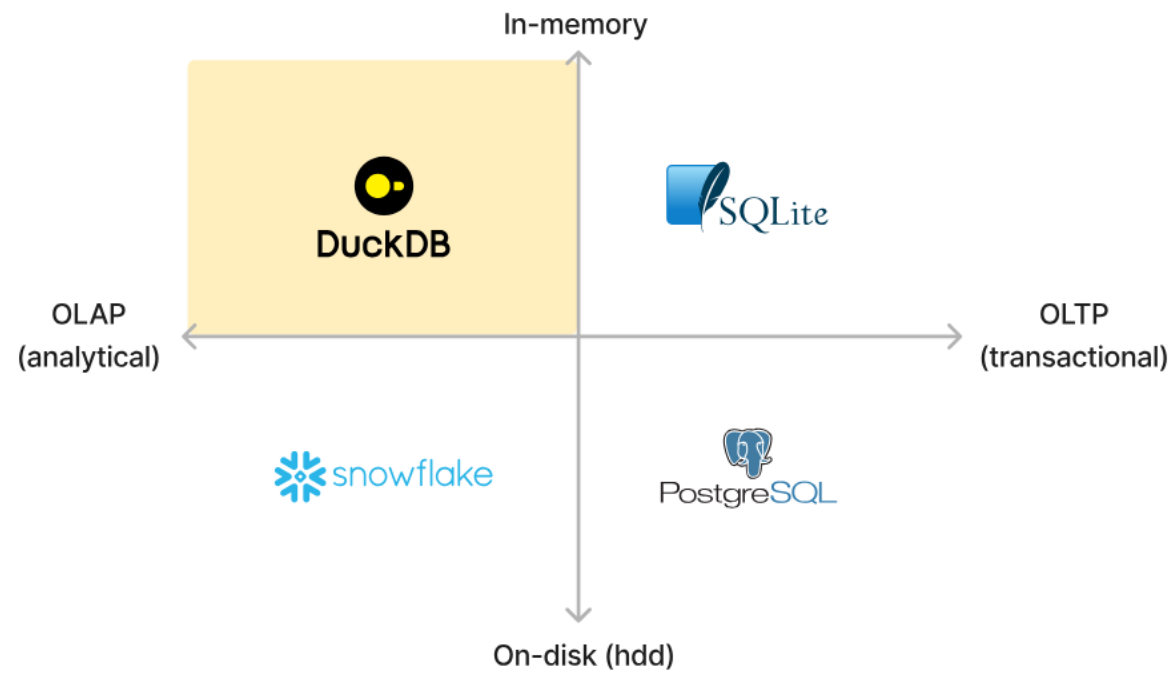
```
1 -- Get the top-3 busiest train stations
2 SELECT
3   station_name,
4   count(*) AS num_services
5 FROM train_services
6 GROUP BY ALL
7 ORDER BY num_services DESC
8 LIMIT 3;
```

Aggregation query ☑

Live demo →

DuckDB

DuckDB



Functions

Functions

If we write the same code in different places and expect it to behave the same everywhere, we will also have to update it in several places whenever we make a change.



This can quickly become unwieldy.



In large codebases, copying code in multiple places would often require us to waste time making the same change in several places.



It would also add the overhead of tracking duplicated code.



Efficiency is the motivation for the **d**on't **r**epeat **y**ourself mantra.

In software engineering, **Don't Repeat Yourself (DRY)** is a principle of software development that we can use functions and modules to avoid repeating code.

<DRY>
DON'T REPEAT YOURSELF

Functions

A function is a block of reusable code that can be used to perform an action.

A function provides better:



Organization

- Functions make code more readable because they wrap blocks of operational code into a single, callable name.



Modularity

- Functions can be called multiple times and used over and over again.
- Functions take in optional inputs and generate output. Function inputs are not bound to a specific variable, but rather a specific data type.



Comprehension

- Functions are often annotated with docstrings, comments that help users understand the specific purpose of a defined function.
- Function names are often good indicators of what the function will be trying to achieve.

Introduction to Functions

Prevent repetition with frequent use of Python functions.

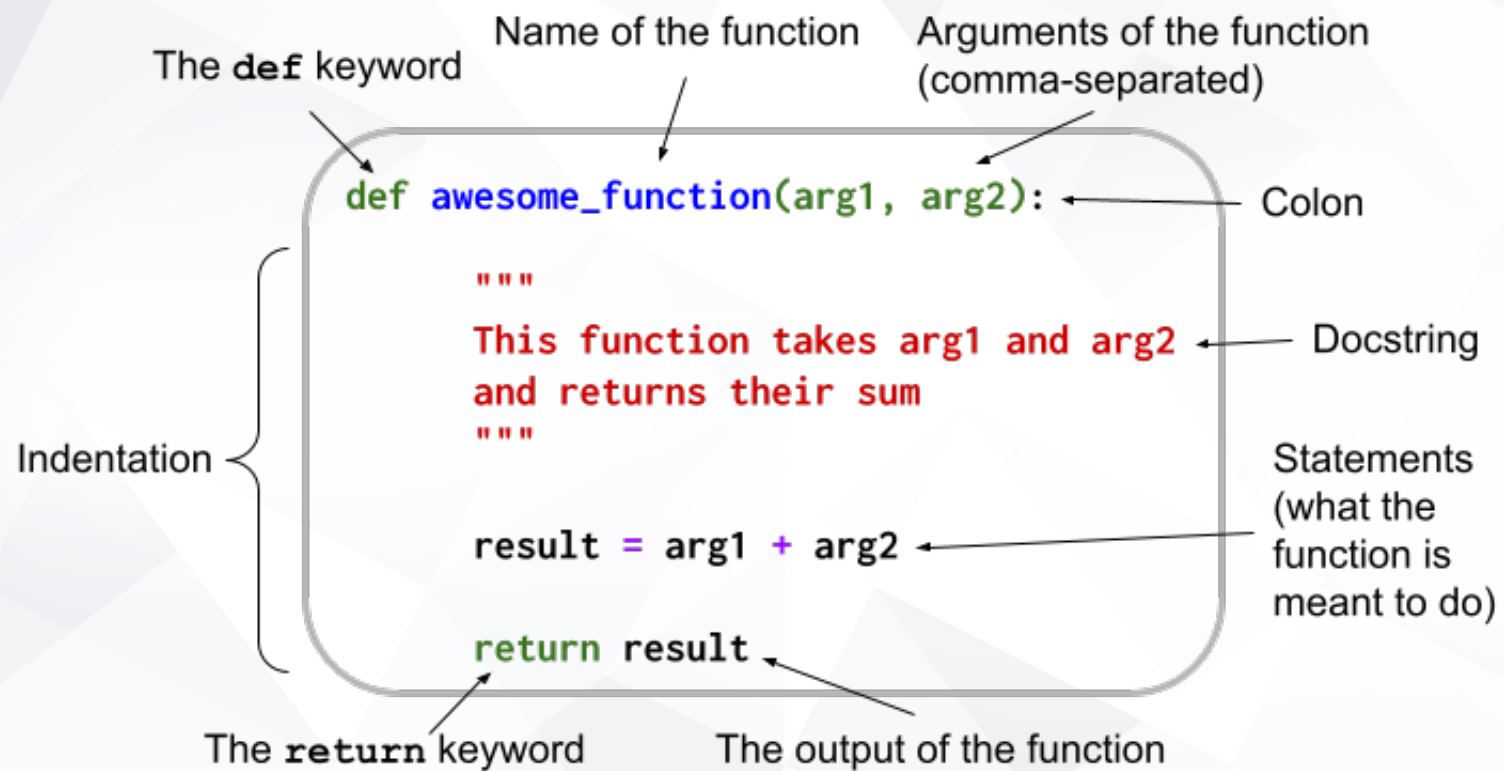
- A function is a block of organized, reusable code that is used to perform a single, related action. In other words, functions are placeable blocks of code that perform a specific action.
- To create a new function, simply use `def` **<FUNCTION NAME>():**, and then place the code that you would like to run within the block underneath it.
- To run the code stored within a function, the function itself must be called within the program. Functions will not run unless called upon.
- Functions that take in parameters can also be created by simply adding a variable into the parentheses of the function's definition. This allows specific data to be passed into the function.

```
def print_hello():  
    print(f"Hello!")  
  
print_hello()
```

```
def print_name(name):  
    print("Hello " + name + "!")  
  
print_name("Bob Smith")
```



Function Anatomy




Function Anatomy

Defining the function, parameters, and return values:

```
1
2
3  def create_phase(name):
4      greeting = f"Hi {name} it's nice to meet
5      you!"
6      return greeting
7
8  sentence = create_phase("Andrew")
9  print(sentence)
```

Function Anatomy

Defining the function, parameters, and return values:




Defines the function `name` as well as the parameters it supports.

```
1
2
3  def create_phase(name):
4      greeting = f"Hi {name} it's nice to meet you!"
5      return greeting
6
7  sentence = create_phase("Andrew")
8  print(sentence)
9
```

Function Anatomy

Defining the function, parameters, and return values:

The function uses
the name parameter
as a substitute for a
dynamically created
greeting string.




```
1
2
3  def create_phase(name):
4      greeting = f"Hi {name} it's nice to meet you!"
5      return greeting
6
7  sentence = create_phase("Andrew")
8  print(sentence)
9
```

Function Anatomy

Defining the function, parameters, and return values:

The function **returns**
the greeting variable.



```
1
2
3  def create_phase(name):
4      greeting = f"Hi {name} it's nice to meet you!"
5      return greeting
6
7  sentence = create_phase("Andrew")
8  print(sentence)
9
```


Function Anatomy

Defining the function, parameters, and return values:

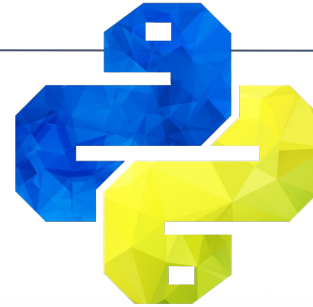
Calls the function
and assigns the
return value to the
variable `sentence`.



```
1
2
3  def create_phase(name):
4      greeting = f"Hi {name} it's nice to meet you!"
5      return greeting
6
7  sentence = create_phase("Andrew")
8  print(sentence)
9
```

Function Anatomy

Showcasing the Python function and output:



```
1
2
3  def create_phase(name):
4      greeting = f"Hi {name} it's nice to meet you!"
5      return greeting
6
7  sentence = create_phase("Andrew")
8  print(sentence)
9
```

```
Desktop — -bash — 80x24
[Administrators-MacBook-Pro:Desktop andrewyang$ python phrase.py]
Hi Andrew it's nice to meet you!
Administrators-MacBook-Pro:Desktop andrewyang$
```

Functions

Calling a function:

```
def calculate_market_cap(market_price, number_of_shares):  
    cap = market_price * number_of_shares  
    return cap  
  
market_price = 76.06  
number_of_shares = 1243600000  
  
market_cap = calculate_market_cap(market_price, number_of_shares)  
print(f"Market Capitalization: {market_cap}")
```

Market Capitalization: 94588216000.0

For Loops

A stylized teal lightning bolt graphic with a 3D effect, pointing towards the text.

**Python can read in data from
external text files and then
perform tasks on that data.**

Reading Text Files

We all need directions to go from point A to point B, and Python is no different when dealing with external files. It requires very precise directions about what path to follow to reach the desired file.

In this case, the desired file is located within a subfolder called “Resources,” so the path we need to provide Python would be “Resources/FileName.txt”.

Note: Different operating systems set their paths in different ways.

```
# Store the file path associated with the file  
(note the backlash may be OS specific)  
file = 'Resources/input.txt'
```

Reading Text Files



`with` is a special syntax block that allows us to perform operations that require a safety clean-up after the code block is completed.



`open<File Path>, <Read/Write>` is the function that Python uses to open a file. By specifying either `'r'`, `'w'`, or `'rw'`, we can read from a text file, write to a text file, or perform both operations.



`text.read()` reads the entire file and converts it to a string type.

```
# Open the file in "read" mode ('r') and store the contents in the variable "text"
with open(file, 'r') as text:

    # Store all of the text inside a variable called "lines"
    lines = text.read()

    # Print the contents of the text file
    print(lines)
```



**Python can read in data from
external text files and then
perform tasks on that data.**

Reading Text Files

We all need directions to go from point A to point B, and Python is no different when dealing with external files. It requires very precise directions about what path to follow to reach the desired file.

In this case, the desired file is located within a subfolder called “Resources,” so the path we need to provide Python would be “Resources/FileName.txt”.

Note: Different operating systems set their paths in different ways.

```
# Store the file path associated with the file  
(note the backlash may be OS specific)  
file = 'Resources/input.txt'
```

Reading Text Files



`with` is a special syntax block that allows us to perform operations that require a safety clean-up after the code block is completed.



`open<File Path>, <Read/Write>` is the function that Python uses to open a file. By specifying either `'r'`, `'w'`, or `'rw'`, we can read from a text file, write to a text file, or perform both operations.



`text.read()` reads the entire file and converts it to a string type.

```
# Open the file in "read" mode ('r') and store the contents in the variable "text"
with open(file, 'r') as text:

    # Store all of the text inside a variable called "lines"
    lines = text.read()

    # Print the contents of the text file
    print(lines)
```



A **for loop** is used for iterating over a sequence such as a list or dictionary.

For Loops

```
# Loop through a range of numbers (0 through 4)
for x in range(5):
    print(x)

print("-----")

# Loop through a range of numbers (2 through 6 - yes 6!, Up to, but not including, 7)
for x in range(2, 7):
    print(x)

print("-----")

# Iterate through letters in a string
word = "Peace"
for letters in word:
    print(letters)

print("-----")
```

Loops

- The variable `x` is created within the loop statement and could theoretically take on any name as long as it is unique.
- When looping through a range of numbers, Python will halt the loop one number before the final number. For example, when looping from 0 to 5, the code will run 5 times, but `x` will only ever be printed as 0 through 4.
- When provided with a single number, `range()` will always start the loop at 0. However, when provided with two numbers, the code will loop from the first number until it reaches one fewer than the second number.

```
# Loop through a range of numbers (0 through 4)
for x in range(5):
    print(x)

print("-----")

# # Loop through a range of numbers (2 through 6)
for x in range(2, 7):
    print(x)

print("-----")
```

Looping through strings

Python can also loop through all the letters within a string.

The syntax is for `<variable> in <string>`:

```
# Iterate through letters in a string
word = "Peace"
for letter in word:
    print(letter)

print("_____")
```

Looping through lists

Python can also loop through all the values within a list.

The syntax is for `<variable> in <list>:`

```
# Iterate through a list
zoo = ['cow', 'dog', 'bee', 'zebra']
for animal in zoo:
    print(animal)

print("_____")
```

Zippping Lists

zip() takes in a series of lists as its parameters and joins them together in a stack.

By zipping these lists together, there is now a single, joined list whose indexes reference all three of the lists inside.

Each zipped object can be used only once. For example, you can write the zipped object to a CSV or print to the terminal, but not both.

```
$ python zipper.py
(1, 'Micheal', 'Boss')
(2, 'Dwight', 'Sales')
(4, 'Meredith', 'Sales')
(4, 'Kelly', 'HR')
```

```
# Three Lists
indexes = [1, 2, 3, 4]
employees = ["Micheal", "Dwight", "Meredith", "Kelly"]
department = ["Boss", "Sales", "Sales", "HR"]

# Zip all three lists together into tuples
roster = zip(indexes, employees, department)

# Print the contents of each row
for employees in roster:
    print(employee)
```


while Loops

These are just like **for** loop but will continue looping for as long as a condition is met.

```
# Loop while a condition is being met
run = 'y'
while run == 'y':
    print('Hi!')
    run = input("To run again. Enter 'y'")
```

What Is a for Loop?

Loops through a range of numbers, the letters in a string, or the elements within a list one by one.

```
jacob@DESKTOP-0ICJMD MINGW64 ~/OneDrive/Documents/WorkAndSchool/TeachingAssistant/DataViz/DataViz-Lesson-Plans/01-Lesson-Plans/03-Python/2/Activities/02-Ins_Simple Loops (Scramble-Branch)
$ python SimpleLoops.py
```

What Is a while Loop?

Loops through the code contained inside of it until some condition is met.

```
jacob@DESKTOP-0ICJMMD MINGW64 ~/OneDrive/Documents/WorkAndSchool/TeachingAssistant/  
DataViz/DataViz-Lesson-Plans/01-Lesson-Plans/03-Python/2/Activities/02-Ins_SimpleLo  
ops (Scramble-Branch)  
$ python SimpleLoops.py
```