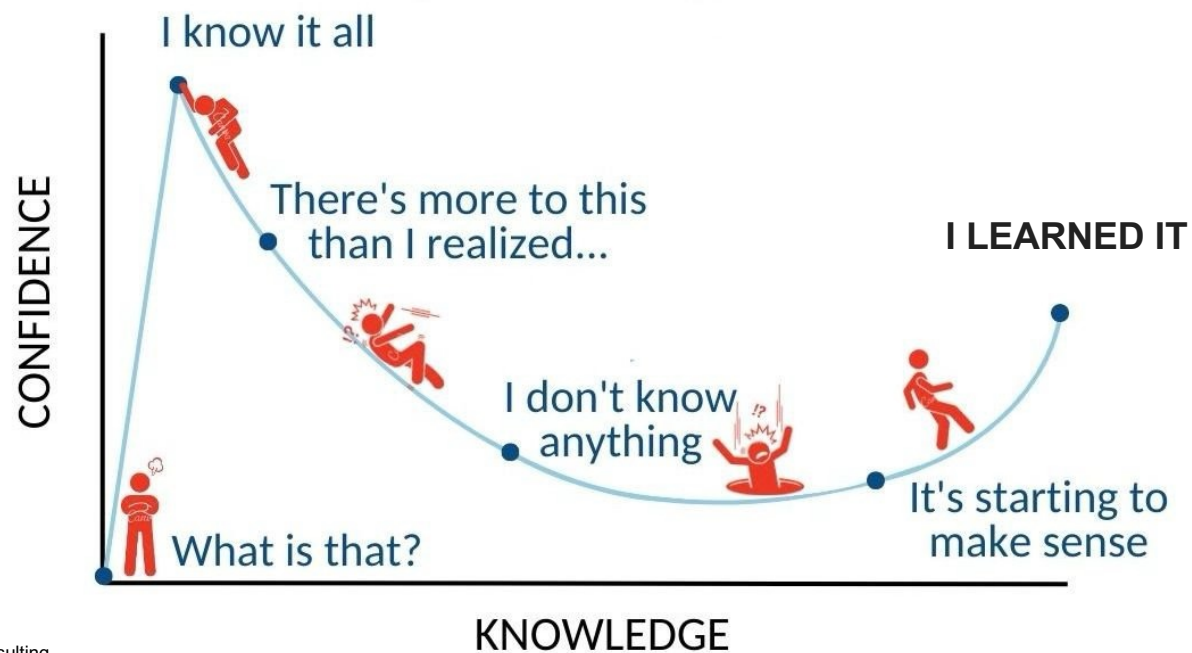


correlation.:one
TECH FOR JOBS

DUNNING-KRUGER Effect




Credit: BVA Nudge Consulting



Python is an interpreted, object-oriented, high-level programming language with dynamic semantics.

Python

Python offers the following features:

Easy-to-Read Syntax	Modularity	Cross-Platform Capability	Robust Modeling, Financial Analytics, and Data Science Platform
Human-readable syntax that makes it easier to understand and debug applications	<p>A wide variety of standard and advanced libraries that can be easily imported into user applications</p> <p>Many well-known data science and visualization libraries such as NumPy, Pandas, and Plotly</p>	Runtime environments can be deployed on Windows, Mac, Linux	<p>Supports a number of financial analytic and data science libraries such as Pandas, NumPy, Matplotlib, SciPy, and SciKit</p> 



Variables are reserved allocations in memory that can hold defined values.

Variables

Variables have three main operations: **create**, **put**, and **retrieve**.

01

To create a variable, **declare** it.

02

To put a value in a variable, **assign** it.

03

To retrieve a variable, **call** it.

Lists

Lists

A list is a data structure with the following characteristics:



It is a collection of elements.



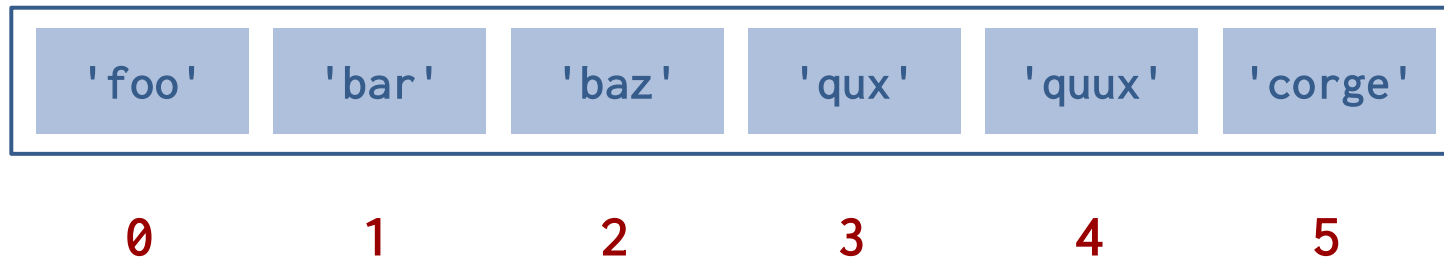
It is ordered.



It is heterogenous (can hold different data types).



It has a zero-based index.





Lists can hold different data types,
**but they commonly hold elements
of a single data type to represent
a conceptual category or group,**
e.g., a list of cars or a group of
people.

List Methods in Python

Python has a set of built-in methods that you can use on lists:

append	method adds elements to the end of a list.
index	method returns the numeric location of a given value within a list.
len	function returns the length of a list.
remove	method deletes a given value from a list.
pop	method can remove a value by index.

```
# Create a variable and set it as an List
myList = ["Jacob", 25, "Ahmed", 80]
print(myList)

# Adds an element onto the end of a List
myList.append("Matt")
print(myList)

# Changes a specified element within an List at the given index
myList[3] = 85
print(myList)

# Returns the index of the first object with a matching value
print(myList.index("Matt"))

# Returns the length of the List
print(len(myList))

# Removes a specified object from an List
myList.remove("Matt")
print(myList)

# Removes the object at the index specified
myList.pop(0)
myList.pop(0)
print(myList)
```

Tuples

Tuples are functionally similar to lists in what they can store, but they are immutable.



While lists in Python can be modified after their creation, tuples can never be modified after their declaration.



Tuples tend to be more efficient to navigate through than lists and also protect the data stored within from being changed.

```
# Creates a tuple, a sequence of immutable Python objects that cannot be changed
myTuple = ('Python', 100, 'VBA', False)
print(myTuple)
```

Dicts

Dicts

Dicts are mutable (changeable), unordered data structures with the following key characteristics:



A collection of key-value pairs



Unordered



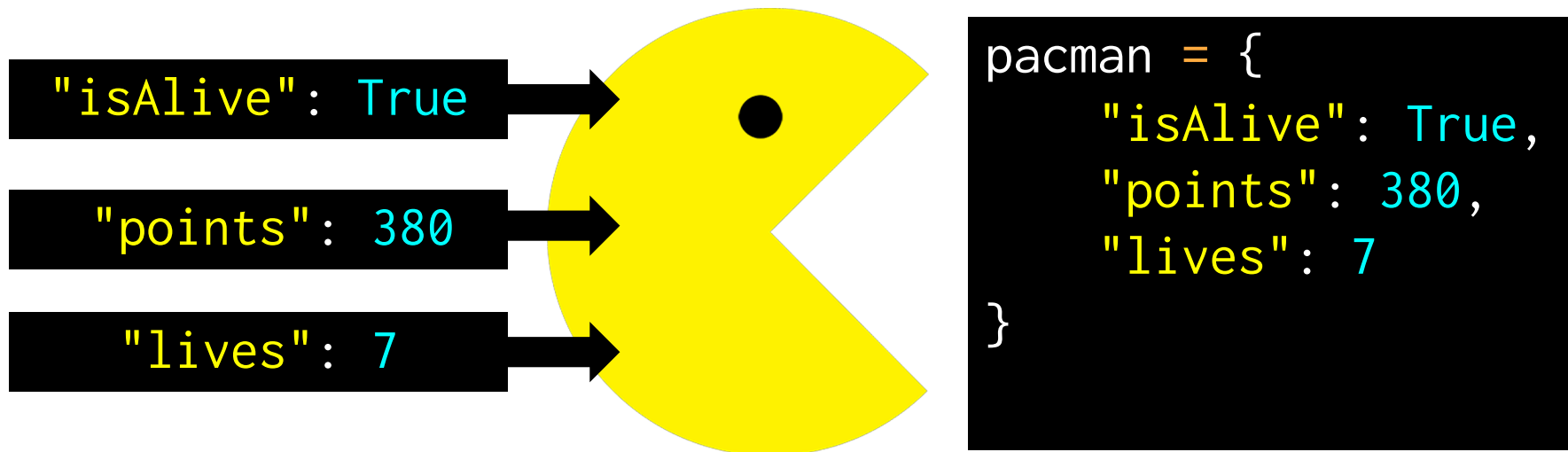
Heterogeneous (can contain different data types)



A **dictionary** is an object that stores a collection of data.

Dictionaries

Like lists and tuples, dictionaries can contain multiple values and data types. However, unlike lists and tuples, dictionaries store data in **key-value** pairs. The key in a dictionary is a string that can be referenced to collect an associated value.



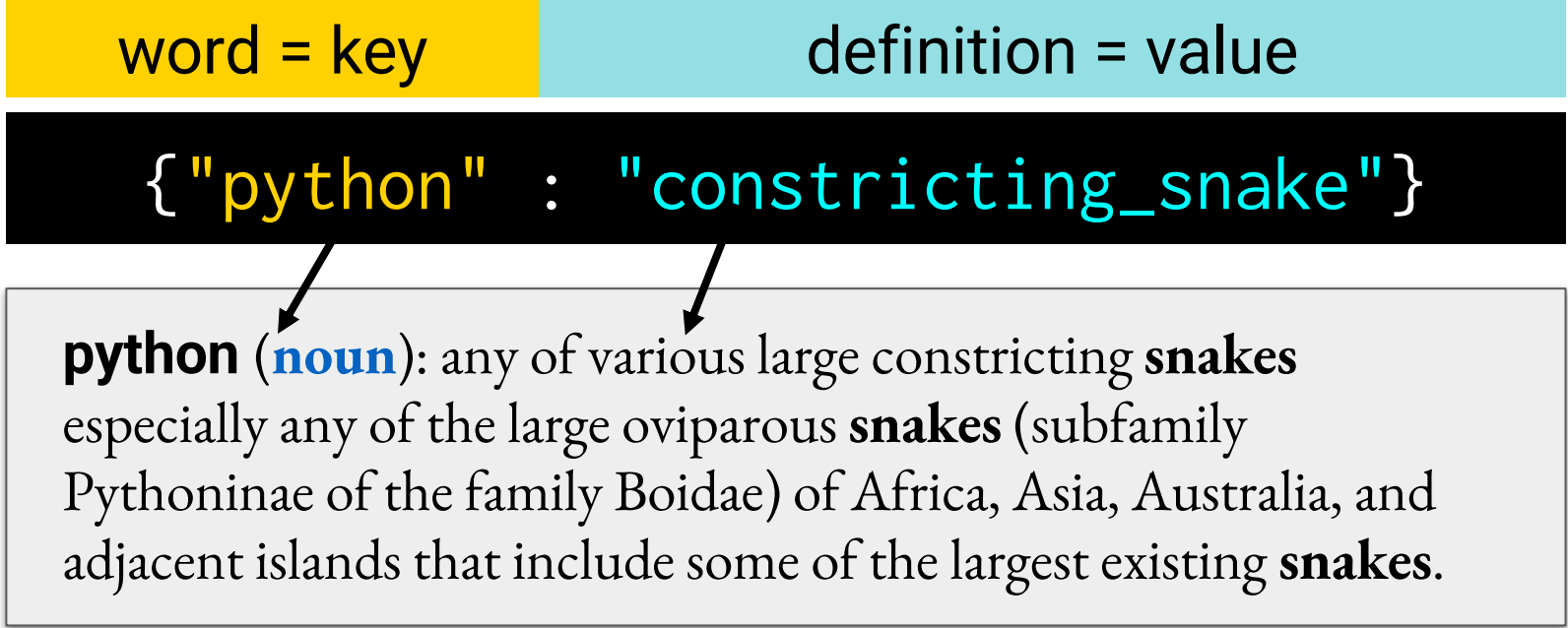
Dictionaries

To use the example of a physical dictionary, the words in the dictionary would be considered the keys, and the definitions of those words would be the values.

word = key

definition = value

```
{"python" : "constricting_snake"}
```



python (**noun**): any of various large constricting **snakes** especially any of the large oviparous **snakes** (subfamily Pythoninae of the family Boidae) of Africa, Asia, Australia, and adjacent islands that include some of the largest existing **snakes**.

Dictionaries

To initialize or create an empty dictionary, we use the syntax `actors = {}`.

```
# Create a dictionary to hold the actor's names.  
actors = {}
```

You can also create a dictionary with the built-in Python `dict()` function, or `actors = dict()`.

```
# Create a dictionary using the built-in function.  
actors = dict()
```

Dictionaries

Values can be added to dictionaries at declaration by creating a key that is stored in a string, following it with a colon, and then placing the desired value after the colon.

To reference a value within a dictionary, we simply call the dictionary and follow it up with a pair of brackets containing the key for the desired value.

```
# A dictionary of an actor.  
actors = {"name": "Tom Cruise"}  
print(f'{actors["name"]}')
```

Dictionaries

Values can also be added to dictionaries by placing the key within single or double quotation marks inside brackets, and then assigning the key a value; then, values can be changed or overwritten by assigning the key a new value.

```
# Add an actor to the dictionary with the key "name"  
# and the value "Denzel Washington".  
actors["name"] = "Denzel Washington"
```

Dictionaries

Dictionaries can hold multiple pieces of information by following up each key-value pairing with a comma and then another key-value pair.

```
# A list of actors
actors_list = [
    "Tom Cruise",
    "Angelina Jolie",
    "Kristen Stewart",
    "Denzel Washington"]

# Overwrite the value, "Tom Cruise", with the list of actors.
actors["name"] = actors_list
```

Dictionaries

Keys

Keys are immutable objects, like integers, floating-point decimals, or strings.

Keys cannot be lists or any other type of mutable object.

Values

Values in a dictionary, as captured in the following image, can be objects of any type:

- integers
- floating-point decimals
- strings, Booleans
- `datetime` values
- lists

Dictionaries

Items in a list in a dictionary can be accessed by calling the key and then using indexing to access the item, as in the following image.

```
# Print the first actor  
print(f'{actors["name"][0]}')
```



**You only need a basic understanding of this for now;
when you get into APIs, you will get a lot more practice!**



**Dictionaries can also contain
other dictionaries.**

Dictionaries

To access the values inside nested dictionaries, simply add another key to the reference.

```
# A dictionary can contain multiple pairs of information
actress = {"name": "Angelina Jolie", "genre": "Action", "nationality": "United States"}

# -----

# A dictionary can contain multiple types of information
another_actor = {"name": "Sylvester Stallone", "age": 62, "married": True, "best movies": "Rocky"}
print(f'{another_actor["name"]} was in {another_actor["best movies"][0]}')

# -----

# A dictionary can even contain another dictionary
film = {"title": "Interstellar",
        "revenues": {"United States": 360, "China": 250, "United Kingdom": 73}}
print(f'{film["title"]} made {film["revenues"]["United States"]} in the US.')

# -----
```


Nested Lists and Dicts

Nested Lists and Dicts

A nested object is an iterable object that contains one or more iterable objects, thereby having “nested” levels of iteration.

Types of nested lists and dicts:



List of lists



List of dicts



Dictionary of lists



Dictionary of dicts

Nested Lists and Dicts

List of lists:

```
# List
ceo_list = ["Warren", "Jack", "Harry"]

# List of Lists
ceo_nested_list = [
    ["Warren Buffet", 88, "CEO of Berkshire Hathaway"],
    ["Jeff Bezos", 55, "CEO of Amazon"],
    ["Harry Markowitz", 91, "Professor of Finance"]
]

# Retrieve first entry of ceo_nested_list
first_entry = ceo_nested_list[0]

# Retrieve name of first entry
first_entry_name = ceo_nested_list[0][0]

# Retrieve age of first entry
first_entry_age = ceo_nested_list[0][1]

# Retrieve occupation of first entry
first_entry_occupation = ceo_nested_list[0][2]

# Print results to screen
print("The first entry in employees_nested_list is:", first_entry)
print(f"{first_entry_name} is {first_entry_age} years old, serving as {first_entry_occupation}.")
```

Nested Lists and Dicts

Dict of lists:

```
# Dictionary of Lists
stocks_nested_list = {
    "APPL": ["Apple", 101.32, "NASDAQ", 937.7],
    "MU": ["Micron Technology", 32.12, "NASDAQ", 48.03],
    "AMD": ["Advanced Micro Devices", 23.12, "NASDAQ", 29.94],
    "TWTR": ["Twitter", 34.40, "NASDAQ", 26.42]
}

# Retrieve entry for APPL
appl_entry = stocks_nested_list["APPL"]

# Retrieve name, stock_price, and exchange for APPL entry
appl_name = stocks_nested_list["APPL"][0]
appl_stock_price = stocks_nested_list["APPL"][1]
appl_exchange = stocks_nested_list["APPL"][2]

# Print results to screen
print(f"APPL ticker stands for {appl_name}. APPL stock price is currently {appl_stock_price}, and
it is available on {appl_exchange}.")
```

Nested Lists and Dicts

Dict of dicts:

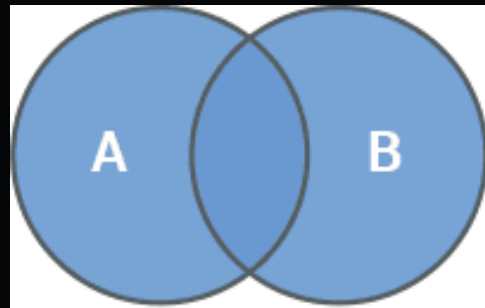
```
# Dictionary of Dicts
stocks_nested_dict = {
    "APPL": {
        "name": "Apple",
        "exchange": "NASDAQ",
        "market_cap": 937.7
    },
    "MU": {
        "name": "Micron Technology",
        "exchange": "NASDAQ",
        "market_cap": 48.03
    },
    "AMD": {
        "name": "Advanced Micro Devices",
        "exchange": "NASDAQ",
        "market_cap": 29.94
    },
    "TWTR": {
        "name": "Twitter",
        "exchange": "NASDAQ",
        "market_cap": 26.42
    }
}

# Retrieve Twitter entry
twitter_entry = stocks_nested_dict["TWTR"]

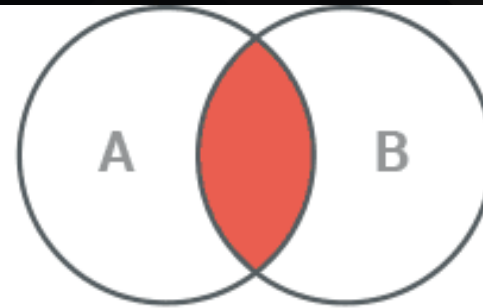
# Retrieve TWTR name, exchange, and market_cap
twitter_name = stocks_nested_dict["TWTR"]["name"]
twitter_exchange = stocks_nested_dict["TWTR"]["exchange"]
twitter_market_cap = stocks_nested_dict["TWTR"]["market_cap"]

# Print results to screen
print(f"Name of TWTR ticker is {twitter_name}. TWTR is available on {twitter_exchange}, and it currently has a market capitalization of {twitter_market_cap}.")
```

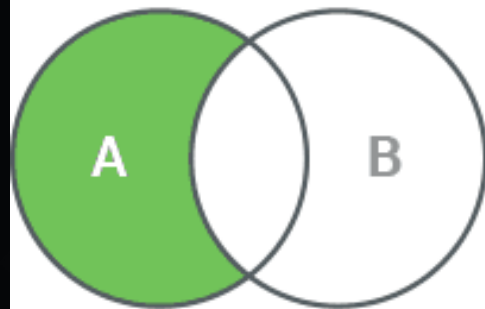
Sets



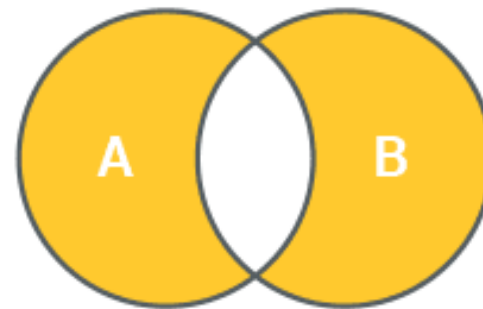
Union



Intersection



Difference



Symmetric Difference