**Key Terms**

* **Variable** - A named location in memory that stores a value. Variables don’t have an explicit type and can be reassigned.
* **Assignment** - The = operator sets a variable name to a value.
* **f-String** - Formatted string literals that allow embedded expressions using f” “ syntax.
* **Integer** - Positive or negative whole number with no decimal point. Supports math operations.
* **Float** - Number with a decimal point. Common for measurements and math with fractions.
* **Boolean** - A True or False value often used for logic.
* **None** - Represents no value assigned. Commonly seen as a default return value.
* **If** - Starts a conditional block that executes if the condition is True.
* **Else** - Adds a block that executes if the prior if condition was False.
* **Exception** - Errors that disrupt normal program flow. Used to catch and handle errors.
* **Try/Except** - Catch exceptions in the except block after first trying the code in the try block.

#print try some of these key terms out

# Variable to store a value

name = "John"

# Reassign variable

name = "Jane"

# f-String prints variable

print(f"Hello {name}")

# Integer

num = 10

# Float

dec = 10.5

# Boolean

is\_true = True

# None

empty = None

# If condition evaluates the Boolean

if is\_true:

  print("Condition was true")

# Else catches when if was False

else:

  print("Condition was false")

# Try block attempts this code

try:

  result = num / 0

# Except catches the ZeroDivisionError

except ZeroDivisionError as e:

  print("Cannot divide by 0")

Output:

Hello Jane

Condition was true

Cannot divide by 0

**Key Terms**

* **List** - An ordered collection of values enclosed in square brackets []. Useful for storing sequences of items.
* **Index** - The numeric position of an item in a list. Starts at 0 for first item. Used to access items by position.
* **Iteration** - Repeated execution of code on successive list items. Done in Python with a for-in loop.
* **Dictionary** - Unordered collection of key-value pairs denoted with curly braces {}. Keys map to associated values.
* **Key** - Unique identifier that is used to look up values in a dictionary. Looks up are very fast.
* **Value** - Data associated with a given key in a dictionary. Values can be any Python data type.
* **Tuple** - Fixed-size, immutable ordered collection similar to a list. Denoted with (). Useful when data shouldn't change.
* **Set** - Unordered collection of unique objects. Helpful for removing duplicates and set operations.
* **Membership** - Ability to check if a value is contained in a collection like lists, dictionaries, tuples, or sets.
* **Methods** - Built-in functions that allow manipulating and interacting with data structures.
* **Iteration** - The process of repeatedly executing code on each item in a collection one by one.

# LIST

nums = [1, 2, 3] # Ordered collection in square brackets

print(nums[0]) # Access first item using INDEX 0

for n in nums: # ITERATE over list

  print(n)

# DICTIONARY

airports = {"SFO": "San Francisco", "LAX": "Los Angeles"} # Key-value pairs in {}

print(airports["SFO"]) # Use KEY to access VALUE

for code in airports:

  print(airports[code]) # VALUE for each KEY

# TUPLE

colors = ("red", "green", "blue") # Immutable ordered collection

print(colors[2])

# SET

unique\_codes = {"SFO", "LAX", "SFO"} # Only 1 SFO

print(len(unique\_codes)) # Length is 2 (no duplicates)

unique\_codes.pop() # Remove item (no index needed)

# MEMBERSHIP check with IN

print("SFO" in airports) # True, has key

# METHOD to add items

unique\_codes.add("ORD")

1

1

2

3

San Francisco

San Francisco

Los Angeles

blue

2

True

# Key Terms

* **loop**: A keyword used for an indefinite loop that runs forever until it is broken out of using a break statement or another control flow mechanism.
* **if let**: A construct allowing assignment and conditional execution based on the result of an expression, such as checking if an optional value contains a specific type (e.g., an integer).
* **shadowing**: The practice of redefining a variable with the same name in the same scope to take on new characteristics or values.
* **type annotations**: Explicitly specifying the data type for variables, expressions, and other elements within Rust code to ensure proper compilation and execution.
* **option**: A wrapper over an optional value that can contain either a concrete value (e.g., integer) or None.
* **sum**: An enum used as a wrapper over an option in the context of this lesson; it provides a convenient way to work with options, especially when combined with pattern matching and if let statements.
* **while loop**: A control flow mechanism that continues executing while a specific condition is met.
* **for loop**: A control flow mechanism used for iterating over sequences (e.g., ranges or vectors) in Rust.
* **continue**: A keyword allowing skipping to the next iteration of a loop when certain conditions are met, such as an odd number in this lesson's example.
* **break**: A keyword that allows breaking out of loops and other control flow mechanisms once specific criteria have been satisfied (e.g., finding a seven).

### Using the loop keyword:

let mut x = 0;

loop {

    if x > 5 {

        break;

    }

    println!("{}", x);

    x += 1; // Adding one to reassign the value of `x`.

}

println!("Loop has ended.");

### Using if let

fn main() {

    let maybe\_number = Some(42); // Wrap the integer 42 in a sum (option) enum.

    if let Some(number) = maybe\_number { // Use pattern matching to extract the value from the option, if it exists.

        println!("The number is: {}", number);

    } else {

        println!("No number was found.");

    }

}

**Key Terms**

* **Append** - Adds an item to the end of a list
* **Insert** - Inserts an item at a specified index in a list
* **Extend** - Appends all items from one list onto another
* **Pop** - Removes and returns an item at a specified index
* **Get** - Safely gets a dictionary value falling back to a default

fruits = ['apple', 'orange']

fruits.append('banana')

print(fruits) # ['apple', 'orange', 'banana']

fruits.insert(0, 'grapes')

print(fruits) # ['grapes', 'apple', 'orange', 'banana']

veggies = ['carrots', 'celery']

fruits.extend(veggies)

print(fruits) # ['grapes', 'apple', 'orange', 'banana', 'carrots', 'celery']

removed = fruits.pop(2)

print(removed) # orange

print(fruits) # ['grapes', 'apple', 'banana', 'carrots', 'celery']

dict = {'name': 'Mary'}

print(dict.get('age', 25)) # 25 (default used)