
MENU
PYTHUNVEIL
HELLO WORLD TO PYTHON
START

# WHAT IS PYTHON

Python is a high-level, general-purpose programming language.

Guido van Rossum, a Dutch programmer, created Python.

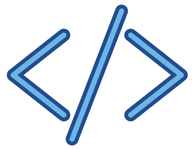
Fun Fact: It is not named after the snake, It is named after the British comedy show "Monty Python's Flying Circus."

# WHAT IS PYTHON

- Python is an interpreted language, not compiled.
- Source code is executed line by line by the Python interpreter.
- This allows for rapid development and easier debugging compared to compiled languages.


LET'S DIVE IN
NOW
EXIT

# AGENDA



BASIC SYNTAX ,VARIABLES AND DATA-TYPES


CONDITIONALS & LOOPS

LISTS,STRINGS, DICTIONARIES

CAPSTONE PROJECT

# BUT FIRST

# DOWNLOAD & SETUP VSCODE

# BASIC SYNTAX ,VARIABLES AND DATA-TYPES

# the print statement

hello.py

```
1    print("hello world")
```

```
print(object(s), sep=separator, end=end, file=file, flush=flush)
```

# BASIC DATA TYPES

| | |
|---|---|
| Text Type: | `str` |
| Numeric Types: | `int`, `float`, `complex` |
| Sequence Types: | `list`, `tuple`, `range` |
| Mapping Type: | `dict` |
| Set Types: | `set`, `frozenset` |
| Boolean Type: | `bool` |

# BASIC DATA TYPES

```
x = 5

print(type(x))
```

# BASIC DATA TYPES

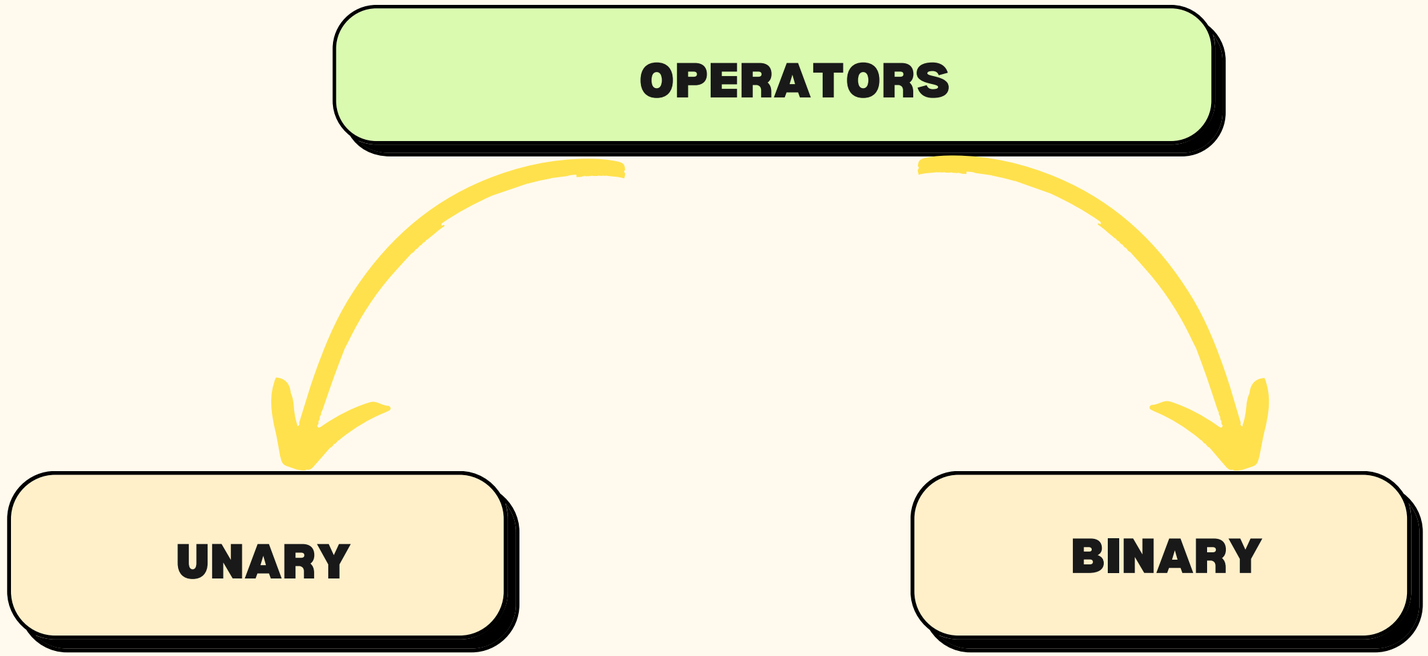
| Example | Data Type |
| --- | --- |
| `x = "Hello World"` | str |
| `x = 20` | int |
| `x = 20.5` | float |
| `x = 1j` | complex |
| `x = ["apple", "banana", "cherry"]` | list |
| `x = ("apple", "banana", "cherry")` | tuple |
| `x = range(6)` | range |
| `x = {"name" : "John", "age" : 36}` | dict |

# VARIABLES

Variables are used to store information to be referenced and manipulated in a computer program

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume). Rules for Python variables:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _ )
- Variable names are case-sensitive (age, Age and AGE are three different variables)
- A variable name cannot be any of the Python keywords.


OPERATORS
UNARY
BINARY

# OPERATORS

Python divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators

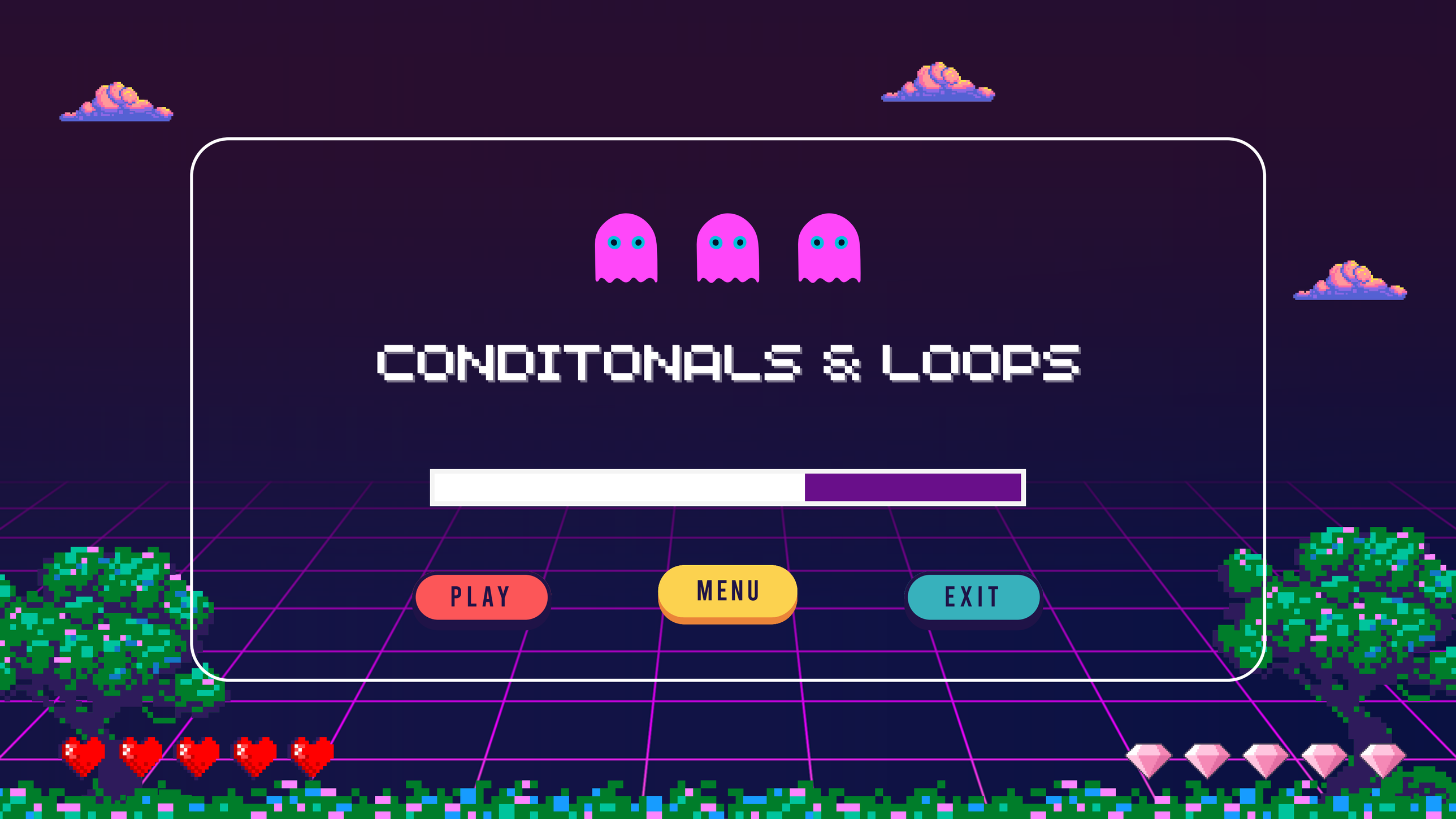
# OPERATORS PRECEDENCE

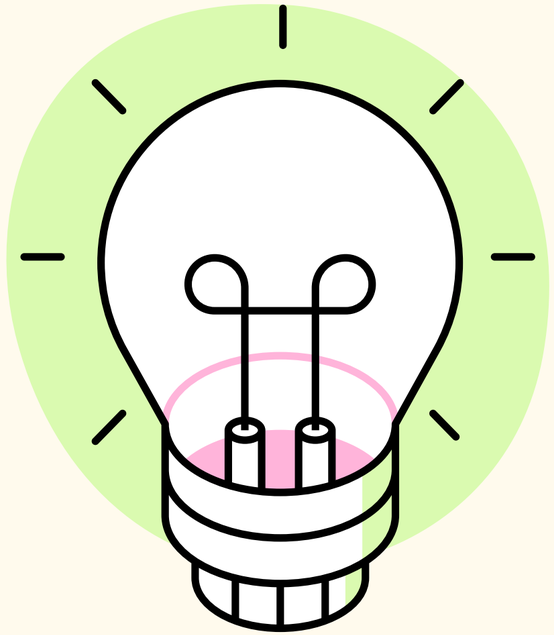
| Operators | Associativity |
|---|---|
| () Highest precedence | Left - Right |
| ** | Right - Left |
| +x , -x, ~x | Left - Right |
| *, /, //, % | Left - Right |
| +, - | Left - Right |
| <<, >> | Left - Right |
| & | Left - Right |
| ^ | Left - Right |
| \| | Left - Right |
| Is, is not, in, not in, <, <=, >, >=, ==, != | Left - Right |
| Not x | Left - Right |
| And | Left - Right |
| Or | Left - Right |
| If else | Left - Right |
| Lambda | Left - Right |
| =, +=, -=, *=, /= Lowest Precedence | Right - Left |

643 × 541

# CHECKPOINT 1 ACHIEVED

EXIT

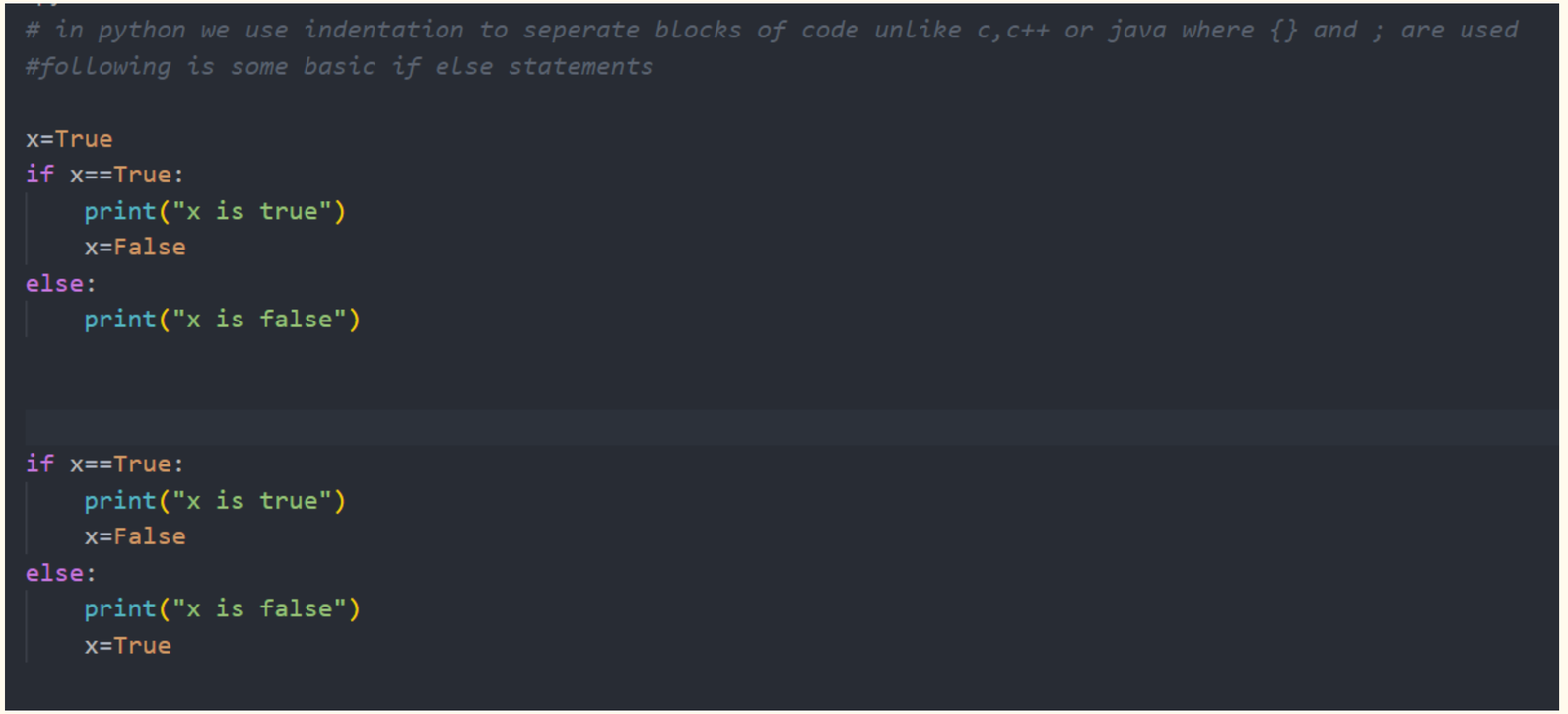

CONDITONALS & LOOPS
PLAY
MENU
EXIT



# WHAT IS IF ELSE

the if else blocks are used to check some logical statement and to trigger somelines of code if it is true

Let's now dive into the syntax of if else



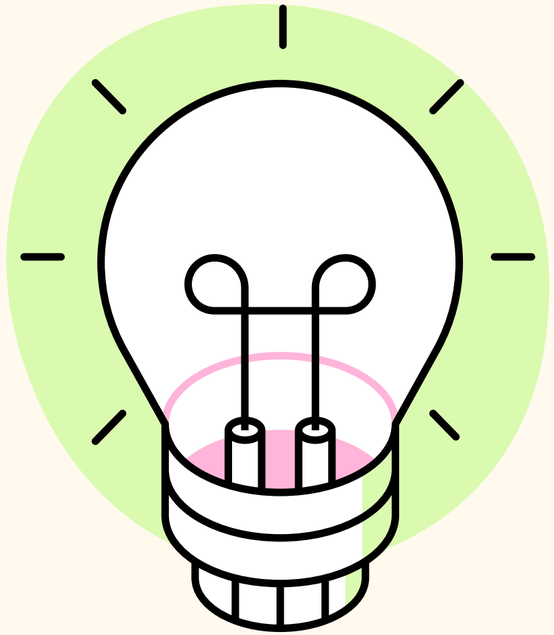
```python
# in python we use indentation to seperate blocks of code unlike c,c++ or java where {} and ; are used
#following is some basic if else statements

x=True
if x==True:
    print("x is true")
    x=False
else:
    print("x is false")



if x==True:
    print("x is true")
    x=False
else:
    print("x is false")
    x=True
```

# A simple if else block

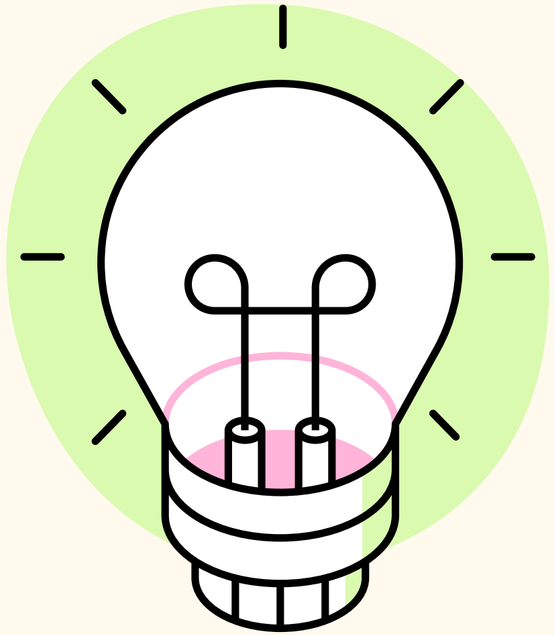


# WHAT IS ELIF?

In Python, the elif statement is short for "else if," and it is used to create a branching structure within conditional statements. It is used in conjunction with if and optionally with else statements to handle multiple conditions in a sequential manner. The elif statement allows you to test multiple conditions one after another and execute specific code blocks based on which condition is true.

```
if condition1:
    # Code to execute if condition1 is true
elif condition2:
    # Code to execute if condition2 is true
elif condition3:
    # Code to execute if condition3 is true
else:
    # Code to execute if none of the conditions are true
```

# Some basic examples with elif



# NESTED IF STATEMENTS

Nested if statements in Python allow you to create more complex branching structures by placing one or more if statements inside another if statement. This is useful when you need to check multiple conditions and have different code blocks executed based on various combinations of those conditions.

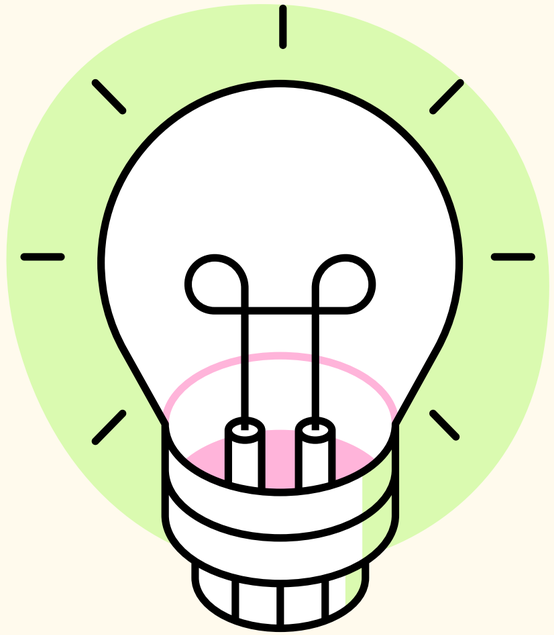
```python
if condition1:
    # Code to execute if condition1 is true

    if nested_condition1:
        # Code to execute if both condition1 and nested_condition1 are true

    elif nested_condition2:
        # Code to execute if condition1 is true and nested_condition2 is true

elif condition2:
    # Code to execute if condition2 is true
```

# Basic Syntax in nested if else



# SHORT HAND IF ELSE

The shorthand "if" and "if-else" expressions in Python, also known as the ternary operator, provide a concise way to handle simple conditions without the need for extensive conditional statements. They follow the format value_if_true if condition else value_if_false, allowing you to assign values to variables based on whether a condition is true or false. This streamlined syntax is useful for compactly expressing straightforward conditions, but it's important to use it thoughtfully to maintain code clarity.

# short hand if syntax

```
if a > b: print("a is greater than b")
```

# short hand if else syntax

```
a = 2
b = 330
print("A") if a > b else print("B")
```

# LOOPS

## WHILE

In Python, a while loop is a control structure that allows you to repeatedly execute a block of code as long as a specified condition remains true.

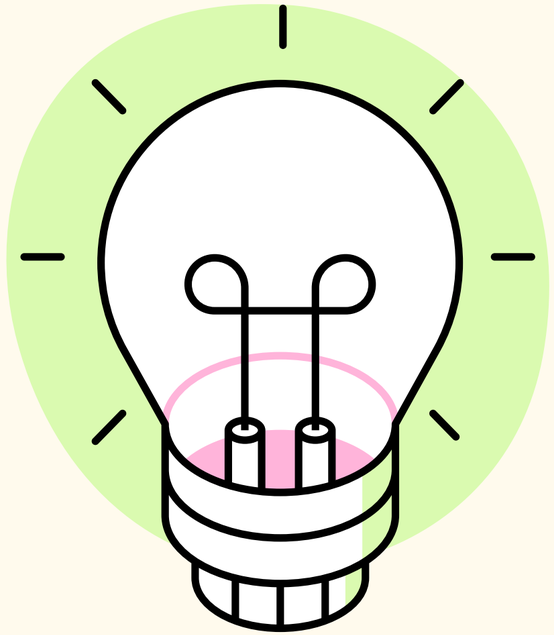
## FOR

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string),also  it can be used when we know how many iterations we need to perform

# WHILE LOOP

```
while condition:
    # Code to execute while the condition is true
```

```
count = 0
while count < 5:
    print("Count:", count)
    count += 1
```



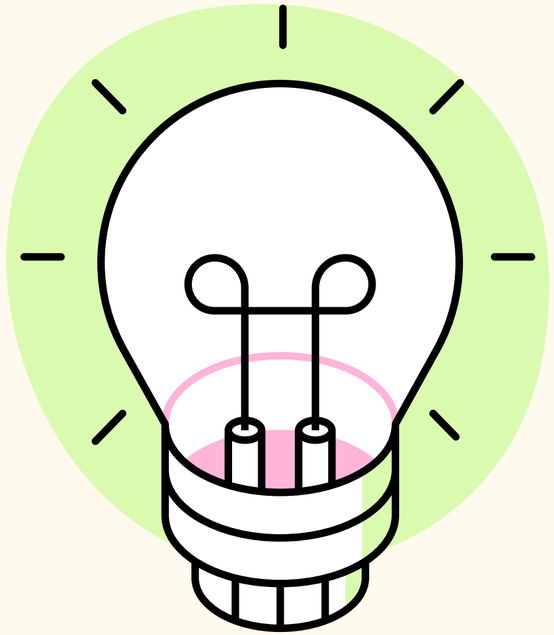
# RANGE FUNCTION

In Python, the range() function is used to generate a sequence of numbers. It is commonly used in for loops to specify the number of iterations or to create a range of values that you want to iterate over. The range() function can take one, two, or three arguments, and it returns an iterable sequence of numbers.

# range function syntax

```
range(stop)
range(start, stop)
range(start, stop, step)
```

- start (optional): The starting value of the sequence. If omitted, the sequence starts from 0.
- stop: The stopping value of the sequence. The sequence includes numbers up to, but not including, this value.
- step (optional): The step or increment between numbers in the sequence. If omitted, the default step is 1.



# FOR LOOP

now that we now about range function, we can now see examples how we can iterate for n times and we will see how to iterate through lists,strings and etc in the next section

# syntax- for loops

```
for item in sequence:
    # Code to execute for each item in the sequence
```

```
for x in range(2, 6):
  print(x)
```

# control statements

# BREAK

The break statement is used to exit the nearest enclosing loop (such as for, while, or do-while) prematurely when a certain condition is met. It allows you to terminate the loop's execution before it would naturally complete.

```
for i in range(1, 6):
    if i == 3:
        break  # Exit the loop when i is 3
    print(i)
```

# control statements

# Continue

The continue statement is used to skip the rest of the current iteration and move to the next iteration of the loop. It is often used when you want to skip specific iterations based on a condition.

```
for i in range(1, 6):
    if i == 3:
        continue  # Skip the iteration when i is 3
    print(i)
```

# control statements
# Pass

The pass statement is a placeholder statement that does nothing. It is used when you need a statement in your code for syntactical reasons but you don't want it to perform any action. It's often used in situations where you're working on code incrementally and want to provide a placeholder for a future implementation.

# CHECKPOINT 2 ACHIEVED

EXIT


BASIC DATA STRUCTURES
LISTS,STRINGS,DICTIO
PLAY
MENU
EXIT

## LISTS

A list in Python is a collection of items enclosed in square brackets []. It can contain a mix of different data types, such as numbers, strings, or even other lists. Lists are ordered and mutable, meaning you can modify their contents after creation. You can access individual items in a list using indexing, starting from 0 for the first item. Lists are versatile and commonly used for storing and managing multiple related values..

## STRINGS

In Python, a string is a sequence of characters enclosed in single ' ' or double " quotes. Strings are used to represent textual data, such as words, sentences, or even symbols. They can be manipulated using various string methods for operations like concatenation, slicing, and formatting. Strings are immutable, which means you can't change their characters directly; instead, you create new strings with the desired modifications.

## DICTIONARIES

A dictionary in Python is a collection of key-value pairs enclosed in curly braces {}. Each key is associated with a value, and you can use the key to access its corresponding value. Dictionaries are unordered and mutable, and they allow you to store and retrieve data using meaningful labels (keys) rather than numerical indices. Dictionaries are great for representing data with a clear relationship between keys and values, and they are often used to store settings, configurations, or structured information.

# Lists syntax and examples

```
numbers = [1, 2, 3, 4, 5]
fruits = ["apple", "banana", "cherry"]
mixed_list = [10, "hello", 3.14, True]

nested_list = [[1, 2, 3], ["a", "b", "c"]]
```

# Accessing lists and list slicing

```
print(numbers[0])   # Output: 1
print(fruits[1])    # Output: "banana"
```

```
subset = numbers[1:4]  # Slice from index 1 to 3 (inclusive)
print(subset)          # Output: [2, 3, 4]
```

# List methods

| Python List Methods | |
|---|---|
| **Method** | **Description** |
| append() | Adds an element at the end of the list |
| clear() | Removes all the elements from the list |
| copy() | Returns a copy of the list |
| count() | Returns the number of elements with the specified value |
| extend() | Add the elements of a list (or any iterable), to the end of the current list |
| index() | Returns the index of the first element with the specified value |
| insert() | Adds an element at the specified position |
| pop() | Removes the element at the specified position |
| remove() | Removes the item with the specified value |
| reverse() | Reverses the order of the list |
| sort() | Sorts the list |

## LISTS

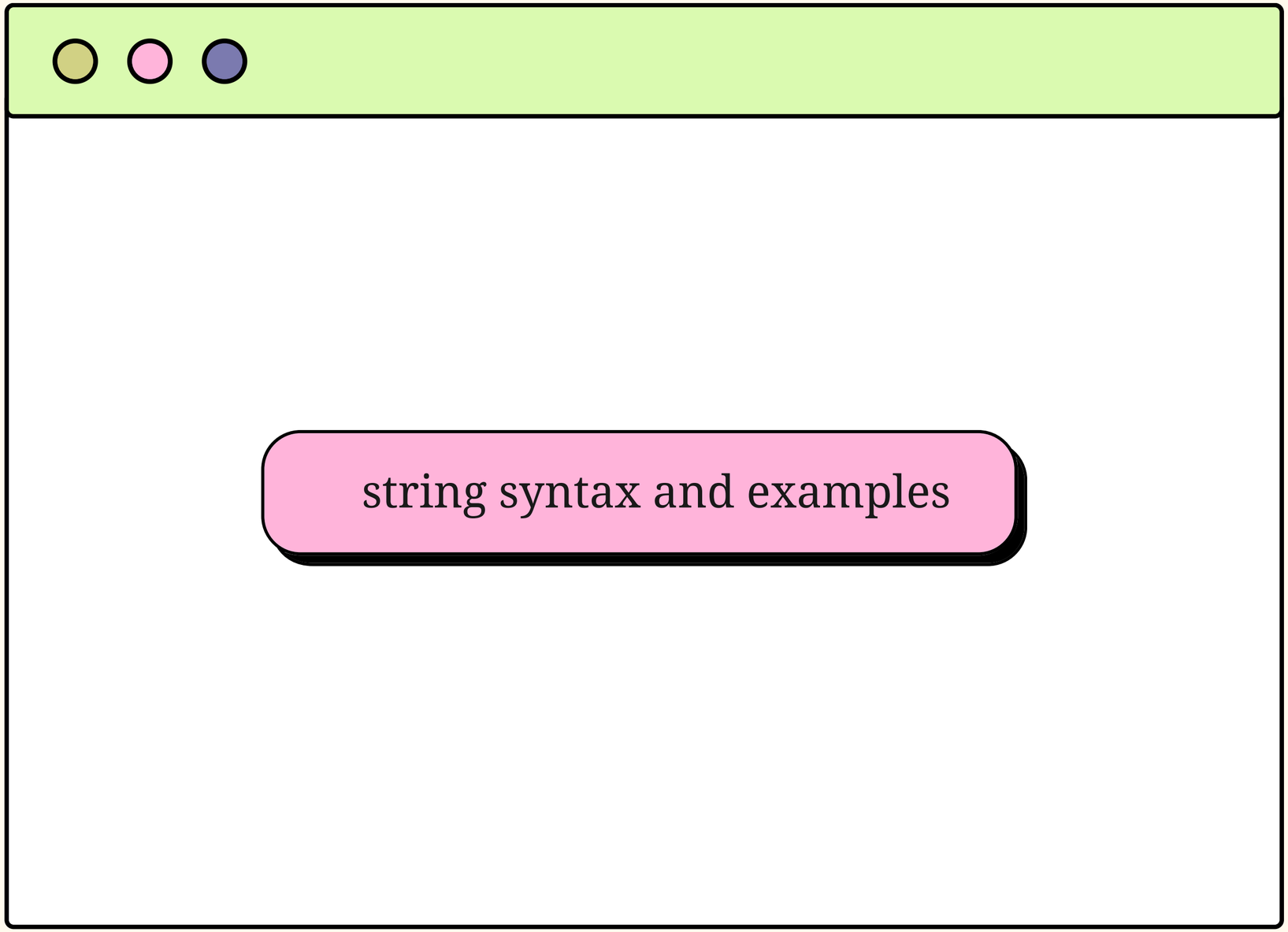
A list in Python is a collection of items enclosed in square brackets []. It can contain a mix of different data types, such as numbers, strings, or even other lists. Lists are ordered and mutable, meaning you can modify their contents after creation. You can access individual items in a list using indexing, starting from 0 for the first item. Lists are versatile and commonly used for storing and managing multiple related values..

## STRINGS

In Python, a string is a sequence of characters enclosed in single ' ' or double " quotes. Strings are used to represent textual data, such as words, sentences, or even symbols. They can be manipulated using various string methods for operations like concatenation, slicing, and formatting. Strings are immutable, which means you can't change their characters directly; instead, you create new strings with the desired modifications.

## DICTIONARIES

A dictionary in Python is a collection of key-value pairs enclosed in curly braces {}. Each key is associated with a value, and you can use the key to access its corresponding value. Dictionaries are unordered and mutable, and they allow you to store and retrieve data using meaningful labels (keys) rather than numerical indices. Dictionaries are great for representing data with a clear relationship between keys and values, and they are often used to store settings, configurations, or structured information.


string syntax and examples

| Method | Description |
| --- | --- |
| capitalize() | Converts the first character to upper case |
| casefold() | Converts string into lower case |
| center() | Returns a centered string |
| count() | Returns the number of times a specified value occurs in a string |
| encode() | Returns an encoded version of the string |
| endswith() | Returns true if the string ends with the specified value |
| expandtabs() | Sets the tab size of the string |
| find() | Searches the string for a specified value and returns the position of where it was found |
| format() | Formats specified values in a string |
| format_map() | Formats specified values in a string |
| index() | Searches the string for a specified value and returns the position of where it was found |
| isalnum() | Returns True if all characters in the string are alphanumeric |
| isalpha() | Returns True if all characters in the string are in the alphabet |
| isascii() | Returns True if all characters in the string are ascii characters |
| isdecimal() | Returns True if all characters in the string are decimals |
| isdigit() | Returns True if all characters in the string are digits |
| isidentifier() | Returns True if the string is an identifier |
| islower() | Returns True if all characters in the string are lower case |
| isnumeric() | Returns True if all characters in the string are numeric |

| | |
|---|---|
| isspace() | Returns True if all characters in the string are whitespaces |
| istitle() | Returns True if the string follows the rules of a title |
| isupper() | Returns True if all characters in the string are upper case |
| join() | Converts the elements of an iterable into a string |
| ljust() | Returns a left justified version of the string |
| lower() | Converts a string into lower case |
| lstrip() | Returns a left trim version of the string |
| maketrans() | Returns a translation table to be used in translations |
| partition() | Returns a tuple where the string is parted into three parts |
| replace() | Returns a string where a specified value is replaced with a specified value |
| rfind() | Searches the string for a specified value and returns the last position of where it was found |
| rindex() | Searches the string for a specified value and returns the last position of where it was found |
| rjust() | Returns a right justified version of the string |
| rpartition() | Returns a tuple where the string is parted into three parts |
| rsplit() | Splits the string at the specified separator, and returns a list |
| rstrip() | Returns a right trim version of the string |
| split() | Splits the string at the specified separator, and returns a list |
| splitlines() | Splits the string at line breaks and returns a list |
| startswith() | Returns true if the string starts with the specified value |
| strip() | Returns a trimmed version of the string |
| swapcase() | Swaps cases, lower case becomes upper case and vice versa |

## LISTS

A list in Python is a collection of items enclosed in square brackets []. It can contain a mix of different data types, such as numbers, strings, or even other lists. Lists are ordered and mutable, meaning you can modify their contents after creation. You can access individual items in a list using indexing, starting from 0 for the first item. Lists are versatile and commonly used for storing and managing multiple related values..

## STRINGS

In Python, a string is a sequence of characters enclosed in single ' ' or double " quotes. Strings are used to represent textual data, such as words, sentences, or even symbols. They can be manipulated using various string methods for operations like concatenation, slicing, and formatting. Strings are immutable, which means you can't change their characters directly; instead, you create new strings with the desired modifications.

## DICTIONARIES

A dictionary in Python is a collection of key-value pairs enclosed in curly braces {}. Each key is associated with a value, and you can use the key to access its corresponding value. Dictionaries are unordered and mutable, and they allow you to store and retrieve data using meaningful labels (keys) rather than numerical indices. Dictionaries are great for representing data with a clear relationship between keys and values, and they are often used to store settings, configurations, or structured information.

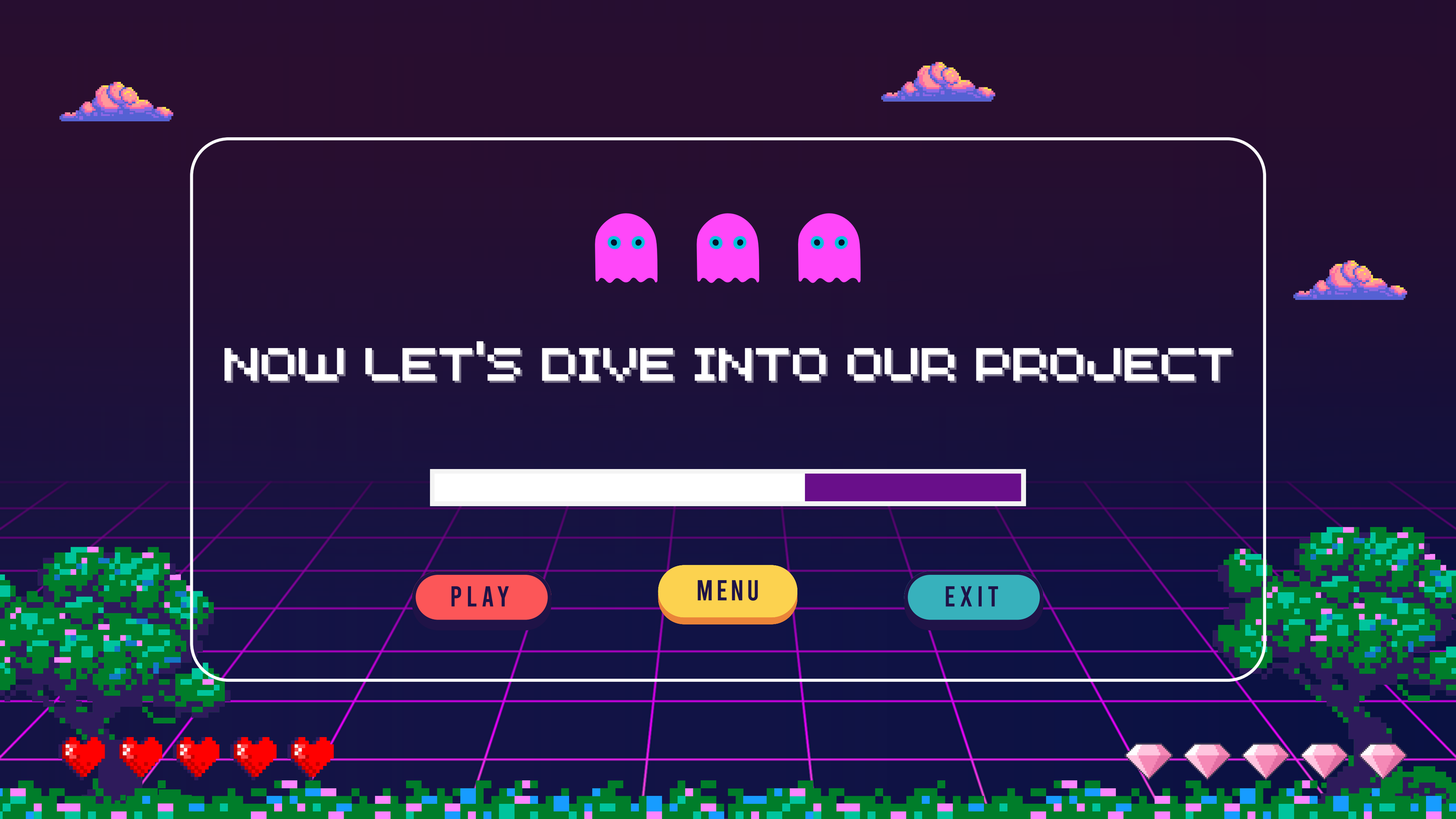
dictionary syntax and examples

# dictionary methods

| Method | Description |
|---|---|
| clear() | Removes all the elements from the dictionary |
| copy() | Returns a copy of the dictionary |
| fromkeys() | Returns a dictionary with the specified keys and value |
| get() | Returns the value of the specified key |
| items() | Returns a list containing a tuple for each key value pair |
| keys() | Returns a list containing the dictionary's keys |
| pop() | Removes the element with the specified key |
| popitem() | Removes the last inserted key-value pair |
| setdefault() | Returns the value of the specified key. If the key does not exist: insert the key, with the specified value |
| update() | Updates the dictionary with the specified key-value pairs |
| values() | Returns a list of all the values in the dictionary |

# CHECKPOINT 3 ACHIEVED

EXIT


NOW LET'S DIVE INTO OUR PROJECT
PLAY
MENU
EXIT