

Python Tutorial for Absolute Beginners

Welcome to Python! This tutorial covers the fundamentals to get you started. We'll explore:

- Variables and basic data types
- The `print()` function (including formatting tricks)
- Basic calculations
- Useful built-in functions (no `if`, `while`, `for`, or defining your own functions)
- **Lists (arrays)** and common operations
- **Tuples** – immutable sequences
- **Sets** – unordered collections of unique elements
- **String methods** like `.split()`, `.join()`, and more
- **File input/output** – reading from and writing to files

Let's dive right in!

1. Your First Python Code

Python is an interpreted language – you can run code line by line. In this tutorial we'll write small snippets that you can try in a Python interpreter (like IDLE, a Jupyter notebook, or an online editor).

```
# This is a comment. It's ignored by Python.  
print("Hello, world!")    # prints text to the screen
```

2. Variables and Basic Data Types

Variables store data. You don't need to declare a type – Python figures it out automatically.

Variable Names

- Can contain letters, digits, and underscores
- Must start with a letter or underscore
- Case-sensitive (`age` and `Age` are different)

Common Data Types

Type	Example	Description
<code>int</code>	<code>42</code> , <code>-3</code> , <code>0</code>	whole numbers
<code>float</code>	<code>3.14</code> , <code>-0.5</code>	numbers with a decimal
<code>str</code>	<code>"hello"</code> , <code>'hi'</code>	text (strings)
<code>bool</code>	<code>True</code> , <code>False</code>	truth values
<code>NoneType</code>	<code>None</code>	represents "nothing"

You can check the type of anything with the `type()` function.

```
# Assign values to variables
name = "Alice"
age = 25
height = 1.68
is_student = True
favourite_color = None

print(name, age, height, is_student, favourite_color)
print(type(name))      # <class 'str'>
print(type(age))       # <class 'int'>
print(type(height))    # <class 'float'>
print(type(is_student))# <class 'bool'>
print(type(favourite_color)) # <class 'NoneType'>
```

3. The `print()` Function – Many Ways to Display Output

`print()` is your window to see results. It can take multiple arguments and has useful features.

Basic printing

```
print("Hello")
print("Hello", "world")  # separates by space by default
print("Hello" + "world") # concatenation (no space)
```

String Concatenation

Combine strings with `+`. Remember to add spaces manually if needed.

```
first = "John"
last = "Doe"
print("My name is " + first + " " + last)
```

f-strings (formatted string literals) – recommended!

Put an `f` before the string and embed variables inside `{}`. This is clean and efficient.

```
name = "Bob"
age = 30
print(f"My name is {name} and I am {age} years old.")
```

You can even do small calculations inside the braces.

```
print(f"Next year I will be {age + 1}.")
```

Using separators and line breaks

The `print()` function accepts a `sep` argument to change the separator.

```
print("apple", "banana", "cherry", sep=", ") # apple, banana, cherry
```

Visual separators

A common trick: repeat a string with `*` to create a dividing line.

```
print("==" * 20) # prints 60 equals signs
```

Use it to separate sections of output:

```
print("Section 1")
print("==" * 20)
print("Section 2")
```

4. Basic Calculations

Python supports all the usual arithmetic operations.

Operator	Meaning	Example	Result
<code>+</code>	addition	<code>10 + 3</code>	<code>13</code>
<code>-</code>	subtraction	<code>10 - 3</code>	<code>7</code>
<code>*</code>	multiplication	<code>10 * 3</code>	<code>30</code>
<code>/</code>	division	<code>10 / 3</code>	<code>3.333...</code>
<code>//</code>	integer division	<code>10 // 3</code>	<code>3</code>
<code>%</code>	remainder (mod)	<code>10 % 3</code>	<code>1</code>

Operator	Meaning	Example	Result
<code>**</code>	exponent	<code>10 ** 3</code>	1000

Examples:

```
a = 15
b = 4

print(f"{a} + {b} = {a + b}")
print(f"{a} - {b} = {a - b}")
print(f"{a} * {b} = {a * b}")
print(f"{a} / {b} = {a / b}")
print(f"{a} // {b} = {a // b}")
print(f"{a} % {b} = {a % b}")
print(f"{a} ** {b} = {a ** b}")
```

Note: mixing integers and floats gives a float result for division (`/`), but `//` gives an integer.

5. Useful Built-in Functions (Without Control Structures)

Python provides many helpful functions that work out of the box. Here are some you can use right away – no `if`, `while`, `for`, or defining your own functions needed.

Type Conversion

Convert between types using functions named after the target type.

- `int()` – converts to integer (truncates decimals, can convert numeric strings)
- `float()` – converts to float
- `str()` – converts to string
- `bool()` – converts to boolean (empty strings, 0, None become `False`; most other things become `True`)

```
price = "19.99"          # string
price_float = float(price) # 19.99 (float)
price_int = int(price_float) # 19 (int)
price_str = str(price_int) # "19" (string)

print(price, type(price))
print(price_float, type(price_float))
print(price_int, type(price_int))
print(price_str, type(price_str))
```

Getting Input from the User

`input()` displays a prompt and returns whatever the user types as a **string**.

```
user_name = input("What is your name? ")
print(f"Hello, {user_name}!")
```

Because `input()` always returns a string, convert it if you need a number.

```
age_str = input("How old are you? ")
age = int(age_str)          # convert to integer
print(f"Next year you will be {age + 1}.")
```

Working with Numbers

- `abs(x)` – absolute value
- `round(x, ndigits)` – rounds a number to a given precision
- `pow(x, y)` – same as `x ** y`
- `divmod(x, y)` – returns both quotient and remainder as a tuple `(x // y, x % y)`

```
print(abs(-7))      # 7
print(round(3.14159, 2))# 3.14
print(pow(2, 5))    # 32
print(divmod(17, 5)) # (3, 2) because 17//5 = 3, 17%5 = 2
```

Finding Length

`len()` returns the number of items in a collection or characters in a string.

```
message = "Hello"
print(len(message))      # 5
```

Minimum, Maximum, Sum

- `min()` – smallest of the given arguments
- `max()` – largest
- `sum()` – sum of an iterable (like a list or tuple). Provide a tuple literal directly.

```
print(min(5, 2, 8, 3))      # 2
print(max(5, 2, 8, 3))      # 8
print(sum((10, 20, 30)))   # 60  (note the double parentheses: sum expects one iterable)
```

You can also sum a list: `sum([10, 20, 30])`.

Other Handy Functions

- `type()` – we already used it
- `help()` – opens the documentation for a function (press `q` to exit)

```
help(print)    # try it!
```

6. Lists (Arrays) in Python

A **list** is an ordered collection of items. You can store anything in a list: numbers, strings, even other lists.

Creating Lists

```
fruits = ["apple", "banana", "cherry"]
mixed = [1, "hello", 3.14, True]
empty = []
```

Accessing Elements

Use square brackets with an **index**. Python indexes start at **0**.

```
print(fruits[0]) # apple
print(fruits[1]) # banana
print(fruits[2]) # cherry
```

Negative indices count from the end:

```
print(fruits[-1]) # cherry (last element)
print(fruits[-2]) # banana
```

List Length

```
print(len(fruits)) # 3
```

Changing Elements

```
fruits[1] = "blueberry"
print(fruits) # ['apple', 'blueberry', 'cherry']
```

Adding Elements

- `.append(item)` – adds an item to the end
- `.insert(index, item)` – inserts at a specific position

```
fruits.append("date")
print(fruits)      # ['apple', 'blueberry', 'cherry', 'date']

fruits.insert(1, "apricot")
print(fruits)      # ['apple', 'apricot', 'blueberry', 'cherry', 'date']
```

Removing Elements

- `.pop()` – removes and returns the last item
- `.pop(index)` – removes and returns the item at that index
- `.remove(item)` – removes the first occurrence of the item

```
last = fruits.pop()
print(last)      # date
print(fruits)    # ['apple', 'apricot', 'blueberry', 'cherry']

fruits.remove("apricot")
print(fruits)    # ['apple', 'blueberry', 'cherry']
```

Slicing Lists

Get a sublist using `[start:end]` (end is exclusive).

```
numbers = [10, 20, 30, 40, 50]
print(numbers[1:4])    # [20, 30, 40]
print(numbers[:3])     # [10, 20, 30] (from beginning)
print(numbers[2:])     # [30, 40, 50] (to the end)
```

List Concatenation and Repetition

```
list1 = [1, 2]
list2 = [3, 4]
combined = list1 + list2    # [1, 2, 3, 4]
repeated = list1 * 3        # [1, 2, 1, 2, 1, 2]
```

Checking if an Item is in a List

```
print("apple" in fruits) # True
print("grape" in fruits) # False
```

7. Tuples – Immutable Sequences

A **tuple** is like a list but **immutable** – once created, you cannot change, add, or remove elements. Tuples are defined using parentheses `()`.

Creating Tuples

```
point = (10, 20)
colors = ("red", "green", "blue")
mixed = (1, "hello", 3.14)
single_element = (42,)          # note the comma – otherwise it's just an int
empty_tuple = ()
```

Accessing Elements

Just like lists, you use indexing and slicing.

```
print(colors[0])      # red
print(colors[-1])     # blue
print(colors[1:3])    # ('green', 'blue')
```

Tuple Length

```
print(len(colors))    # 3
```

Why Use Tuples?

- They are faster than lists
- They protect data that shouldn't change
- They can be used as keys in dictionaries (unlike lists)

Tuple Methods

Tuples have only two methods: `.count()` and `.index()`.

```
numbers = (1, 2, 3, 2, 4, 2)
print(numbers.count(2))      # 3
print(numbers.index(3))      # 2 (first occurrence of 3)
```

Concatenation and Repetition

```
tuple1 = (1, 2)
tuple2 = (3, 4)
combined = tuple1 + tuple2    # (1, 2, 3, 4)
repeated = tuple1 * 3        # (1, 2, 1, 2, 1, 2)
```

Checking Membership

```
print(2 in numbers)          # True
print(5 in numbers)          # False
```

8. Sets – Unordered Collections of Unique Elements

A **set** is an unordered collection of **unique** elements. Sets are defined using curly braces `{}` or the `set()` constructor. They are useful for removing duplicates and performing mathematical set operations.

Creating Sets

```
fruits = {"apple", "banana", "cherry"}  
mixed = {1, "hello", 3.14}  
empty_set = set()           # note: {} creates an empty dict, not a set
```

Set Characteristics

- **Unordered** – you cannot access elements by index
- **No duplicates** – adding a duplicate has no effect

```
numbers = {1, 2, 2, 3, 4, 4}  
print(numbers)           # {1, 2, 3, 4}
```

Adding Elements

- `.add(item)` – adds a single element
- `.update(iterable)` – adds multiple elements (from a list, tuple, etc.)

```
fruits.add("orange")  
print(fruits)           # {'apple', 'banana', 'cherry', 'orange'}  
  
fruits.update(["mango", "grape"])  
print(fruits)           # {'apple', 'banana', 'cherry', 'orange', 'mango', 'grape'}
```

Removing Elements

- `.remove(item)` – removes the item; raises an error if not present
- `.discard(item)` – removes the item if present; does nothing if not present
- `.pop()` – removes and returns an arbitrary element (sets are unordered)
- `.clear()` – removes all elements

```

fruits.remove("banana")
print(fruits)           # {'apple', 'cherry', 'orange', 'mango', 'grape'}

fruits.discard("kiwi")      # no error, even though "kiwi" isn't there

popped = fruits.pop()       # removes some element
print(popped)
print(fruits)              # remaining set

```

Set Operations

Sets support mathematical operations like union, intersection, difference, and symmetric difference. These operations return new sets.

Operation	Method	Operator	Description
Union	<code>set1.union(set2)</code>	<code>set1 \ set2</code>	all elements from both sets
Intersection	<code>set1.intersection(set2)</code>	<code>set1 & set2</code>	elements common to both sets
Difference	<code>set1.difference(set2)</code>	<code>set1 - set2</code>	elements in set1 but not in set2
Symmetric Difference	<code>set1.symmetric_difference(set2)</code>	<code>set1 ^ set2</code>	elements in either set, but not both

```

A = {1, 2, 3, 4}
B = {3, 4, 5, 6}

print(A | B)          # {1, 2, 3, 4, 5, 6}
print(A & B)          # {3, 4}
print(A - B)          # {1, 2}
print(B - A)          # {5, 6}
print(A ^ B)          # {1, 2, 5, 6}

```

Checking Subset / Superset

- `.issubset(other)` – `True` if all elements of set are in other
- `.issuperset(other)` – `True` if all elements of other are in set

```
X = {1, 2}
Y = {1, 2, 3}
print(X.issubset(Y)) # True
print(Y.issuperset(X)) # True
```

Checking Membership

```
print(2 in A)      # True
print(7 in A)      # False
```

9. Useful String Methods

Strings have many built-in methods that return new strings (strings are **immutable** – they cannot be changed, but methods give you a modified copy).

Splitting a String into a List - `.split()`

```
sentence = "Python is awesome"
words = sentence.split()      # splits on whitespace by default
print(words)                  # ['Python', 'is', 'awesome']

csv = "apple,banana,cherry"
items = csv.split(",")        # split on comma
print(items)                  # ['apple', 'banana', 'cherry']
```

Joining a List into a String - `.join()`

```
words = ['Hello', 'world']
sentence = " ".join(words)      # join with a space
print(sentence)                 # Hello world

tags = ['python', 'beginner', 'tutorial']
hashtag = "#".join(tags)
print("#" + hashtag)           # #python#beginner#tutorial
```

Changing Case

```
text = "Hello World"
print(text.upper())            # HELLO WORLD
print(text.lower())            # hello world
print(text.title())            # Hello World
print(text.swapcase())         # hELLO wORLD
```

Stripping Whitespace

```
messy = " \t hello \n "
clean = messy.strip()          # removes leading/trailing whitespace
print(clean)                  # hello
```

Replacing Substrings

```
text = "I like cats"
new_text = text.replace("cats", "dogs")
print(new_text)                # I like dogs
```

Finding Substrings

```
text = "hello world"
print(text.find("world"))      # 6 (index where "world" starts)
print(text.find("xyz"))       # -1 (not found)
```

Checking Start/End

```
filename = "image.jpg"
print(filename.endswith(".jpg"))  # True
print(filename.startswith("img")) # False
```

Counting Occurrences

```
sentence = "how much wood would a woodchuck chuck"
print(sentence.count("wood"))    # 2
```

10. File Input/Output

You can read from and write to files using Python's built-in `open()` function.

Opening and Closing Files

Always close a file after you're done. A safer way is to use a `with` statement, which automatically closes the file.

```
# Writing to a file
with open("output.txt", "w") as file:
    file.write("Hello, file!\n")
    file.write("This is a second line.")
# File is automatically closed when the 'with' block ends

# Reading from a file
with open("output.txt", "r") as file:
    content = file.read()          # reads entire file as one string
    print(content)
```

Reading Line by Line

`.readlines()` returns a list of strings, each representing one line (with the newline character included).

```
with open("output.txt", "r") as file:
    lines = file.readlines()
    print(lines)                  # ['Hello, file!\n', 'This is a second line.']
    # You can access individual lines by index
    print(lines[0])              # Hello, file!
    print(lines[1])              # This is a second line.
```

You can also use `.readline()` to read one line at a time, but without loops it's limited.

```
with open("output.txt", "r") as file:
    first_line = file.readline()
    second_line = file.readline()
    print(first_line, end="")      # Hello, file!
    print(second_line, end="")      # This is a second line.
```

Appending to a File

Use mode `"a"` to add content to the end without overwriting.

```
with open("output.txt", "a") as file:
    file.write("\nThis line is appended.")
```

Writing Lists to a File

If you have a list of strings, you can join them and write.

```
lines_to_write = ["First line", "Second line", "Third line"]
with open("mydata.txt", "w") as file:
    file.write("\n".join(lines_to_write))
```

Reading a File into a List (one line per element)

```
with open("mydata.txt", "r") as file:
    data = file.read().splitlines() # splits on newlines, returns list without trailing \n
    print(data) # ['First line', 'Second line', 'Third line']
```

Checking if a File Exists (without if)

We can use the `os.path` module, but that may be too advanced. However, we can just try to open and handle errors? That would require try/except, which is beyond our scope (no if/else). For absolute beginners, it's okay to assume the file exists. We'll keep it simple.

11. Putting It All Together – A Mini Program

Let's combine lists, tuples, sets, string methods, and file I/O into a small program. This program asks the user for a list of words, removes duplicates using a set, sorts them (but we can't sort without functions? Actually `sorted()` is a built-in function that returns a new list, and we can use it without loops), and writes them to a file.

```
print("==" * 20)
print("WORD PROCESSOR")
print("==" * 20)

# Get words from the user (without loops, just repeated input)
word1 = input("Enter word #1: ")
word2 = input("Enter word #2: ")
word3 = input("Enter word #3: ")
word4 = input("Enter word #4: ")
```

```

word5 = input("Enter word #5: ")

# Put them in a list
word_list = [word1, word2, word3, word4, word5]

# Remove duplicates by converting to a set and back to a list
unique_words = list(set(word_list))
print(f"\nYou entered {len(word_list)} words, but only {len(unique_words)} are unique.")

# Sort the unique words (sorted() returns a new sorted list)
sorted_words = sorted(unique_words)

# Display the sorted unique words
print("\nSorted unique words:")
print(f"1. {sorted_words[0]}" if len(sorted_words) > 0 else "")
print(f"2. {sorted_words[1]}" if len(sorted_words) > 1 else "")
print(f"3. {sorted_words[2]}" if len(sorted_words) > 2 else "")
print(f"4. {sorted_words[3]}" if len(sorted_words) > 3 else "")
print(f"5. {sorted_words[4]}" if len(sorted_words) > 4 else "")

# Write them to a file
with open("words.txt", "w") as f:
    f.write("\n".join(sorted_words))

print("\nWords saved to 'words.txt'.")

# Read the file back and display its content as a tuple of lines
with open("words.txt", "r") as f:
    content_tuple = tuple(f.read().splitlines())

print(f"\nContent read from file (as a tuple): {content_tuple}")

```

This example demonstrates:

- Creating a list from user input
- Using a set to remove duplicates
- Using `sorted()` (built-in) to sort
- Writing to a file with `join()`

- Reading back and converting to a tuple
-

12. Summary

In this extended tutorial you learned:

- **Variables and basic data types** (`int`, `float`, `str`, `bool`, `None`)
- **Print variations** – f-strings, concatenation, separators, and the `"=="*20` trick
- **Basic arithmetic** operations
- **Useful built-in functions** like `type()`, `int()`, `float()`, `input()`, `abs()`, `round()`, `min()`, `max()`, `sum()`, `len()`, `sorted()`
- **Lists (arrays)** – creating, indexing, slicing, modifying, and basic methods
- **Tuples** – immutable sequences, indexing, slicing, `.count()`, `.index()`
- **Sets** – unordered collections of unique elements, adding/removing, set operations (union, intersection, etc.)
- **String methods** – `.split()`, `.join()`, `.upper()`, `.lower()`, `.strip()`, `.replace()`, `.find()`, and more
- **File input/output** – reading and writing files using `open()`, `with`, `.read()`, `.readlines()`, `.write()`

All of this was presented **without using `if`, `while`, `for`, or defining your own functions**, keeping it accessible for absolute beginners.

Now you have the building blocks to write simple but useful Python programs. Experiment with these examples, modify them, and see what happens. Happy coding!

Python Practice Questions – Beginner Level

This document contains practice questions for each section of the **Python Tutorial for Absolute Beginners**. Use these to test your understanding. Try to answer without looking at the tutorial first, then check your answers at the end.

Section 1: Your First Python Code

1. What will the following code print?

```
print("Hello, world!")
```

2. Which symbol is used to write a comment in Python?
 3. Write a single line of Python code that prints your name.
-

Section 2: Variables and Basic Data Types

1. Which of the following are **valid** variable names in Python?

- a) `2nd_value`
- b) `my_var`
- c) `my-var`
- d) `_secret`

2. What is the data type of each value?

- a) `42`
- b) `3.14`
- c) `"Python"`
- d) `True`
- e) `None`

3. Predict the output:

```
a = 10
b = "10"
print(type(a))
print(type(b))
```

4. Write code that creates a variable `age` with the value 25, then prints its type.
 5. What is the result of `bool(0)`? What about `bool("")`?
-

Section 3: The `print()` Function

1. What will the following code display?

```
print("One", "Two", "Three")
```

2. Write a print statement that outputs `Hello` and `world` with a comma and a space between them (like `Hello, world`).

3. Given `name = "Alice"` and `age = 30`, write an f-string that prints:

`Alice is 30 years old.`

4. What does `print("==" * 10)` produce?

5. Write a print statement that uses the `sep` argument to print the numbers 1, 2, 3 separated by a dash (`-`).

Section 4: Basic Calculations

1. Evaluate the following expressions (without running the code):

- o a) `15 + 3 * 2`
- o b) `(15 + 3) * 2`
- o c) `20 // 6`
- o d) `20 / 6`
- o e) `2 ** 4`
- o f) `17 % 5`

2. Write a Python expression that calculates the area of a rectangle with width `w = 5.5` and height `h = 3`. Store the result in a variable and print it.

3. What is the difference between `/` and `//` in Python?

Section 5: Useful Built-in Functions

1. What will `abs(-15)` return?

2. What does `round(3.14159, 2)` give?

3. Use `divmod(20, 3)` – what are the two numbers returned?
 4. Write code that asks the user for their age (as a string), converts it to an integer, and prints how old they will be next year.
 5. What is the output of `len("Hello world")` ?
 6. Given `numbers = [5, 2, 8, 1, 9]`, write expressions using `min()`, `max()`, and `sum()` to find:
 - the smallest number
 - the largest number
 - the sum of all numbers
 7. What does `type(print)` return? (Think carefully – `print` itself is a function.)
-

Section 6: Lists

1. Create a list named `colors` containing the strings `"red"`, `"green"`, and `"blue"`.
 2. From the list `colors`, how do you access the second element? What index do you use?
 3. What will `print(colors[-1])` output?
 4. Add the color `"yellow"` to the end of the list using an appropriate method.
 5. Insert `"purple"` at the beginning of the list (index 0).
 6. Remove and return the last element of the list. What method do you use?
 7. What does `len(colors)` give after you've done the above steps? (Assume you started with 3 colors, added two, then removed one.)
 8. Given `nums = [10, 20, 30, 40, 50]`, write a slice that gives `[20, 30, 40]`.
 9. What is the result of `[1, 2] + [3, 4]`? What about `[1, 2] * 3`?
 10. Check if `"green"` is in the `colors` list. Write the expression.
-

Section 7: Tuples

1. How do you create a tuple containing the numbers 5, 10, 15?
2. What is the difference between a tuple and a list?
3. Can you change an element of a tuple after it's created? Why or why not?

- Given `t = (1, 2, 3, 2, 4, 2)`, what does `t.count(2)` return? What does `t.index(3)` return?
 - Write a tuple with a single element, the string `"alone"`. (Make sure it's a tuple, not just a string.)
 - What will `(1, 2) + (3, 4)` produce?
 - Can you use `in` to check if an element exists in a tuple? Give an example.
-

Section 8: Sets

- Create a set named `letters` containing `'a'`, `'b'`, `'c'`.
- What will happen if you create a set with duplicate values, like `{1, 2, 2, 3}`?
- Add the element `'d'` to the set `letters`.
- Remove `'b'` from the set using a method that will **not** cause an error if `'b'` is missing. Which method do you use?
- Given `A = {1, 2, 3, 4}` and `B = {3, 4, 5, 6}`, find:
 - the union of A and B
 - the intersection of A and B
 - the difference $A - B$
 - the symmetric difference

Write the Python expressions using both methods and operators.

- Check if `2` is a member of the set `A`.
 - What is the output of `len({10, 20, 30, 10, 20})`?
 - Can you access an element in a set by index? Why?
-

Section 9: Useful String Methods

- Given `s = "Python programming is fun"`, what does `s.split()` return?
- Write code that takes the list `words = ['Hello', 'world']` and joins them into a single string with a space between.
- Convert the string `"hello"` to uppercase.
- Remove all leading and trailing whitespace from `"\n text \t "`.

5. Replace "cats" with "dogs" in the string "I love cats".
 6. Find the index of the substring "pro" in "Python programming". What does `find` return if the substring is not found?
 7. Check if the string "photo.jpg" ends with ".jpg".
 8. Count how many times the letter 'p' appears in "apple pie".
-

Section 10: File Input/Output

1. Write code to open a file named `test.txt` in write mode and write the line "Hello, file!" to it. Use a `with` statement.
 2. How would you read the entire content of `test.txt` as a single string?
 3. What does `readlines()` return when used on a file object?
 4. Write code that appends the line "Another line" to the file `test.txt` without overwriting existing content.
 5. Suppose you have a list `lines = ['first', 'second', 'third']`. Write code that writes each element of `lines` to a file `data.txt`, one per line.
 6. How can you read a file into a list where each element is a line from the file, but without the newline characters at the end?
-

Section 11: Putting It All Together – Mini Program

The following program was presented in the tutorial. Identify the concepts used in each part.

```
print("==" * 20)
print("    WORD PROCESSOR")
print("==" * 20)

word1 = input("Enter word #1: ")
word2 = input("Enter word #2: ")
word3 = input("Enter word #3: ")
word4 = input("Enter word #4: ")
word5 = input("Enter word #5: ")

word_list = [word1, word2, word3, word4, word5]
unique_words = list(set(word_list))
print(f"\nYou entered {len(word_list)} words, but only {len(unique_words)} are unique.")
```

```
sorted_words = sorted(unique_words)

print("\nsorted unique words:")
print(f"1. {sorted_words[0]}" if len(sorted_words) > 0 else "")
print(f"2. {sorted_words[1]}" if len(sorted_words) > 1 else "")
print(f"3. {sorted_words[2]}" if len(sorted_words) > 2 else "")
print(f"4. {sorted_words[3]}" if len(sorted_words) > 3 else "")
print(f"5. {sorted_words[4]}" if len(sorted_words) > 4 else "")

with open("words.txt", "w") as f:
    f.write("\n".join(sorted_words))

print("\nwords saved to 'words.txt'.")

with open("words.txt", "r") as f:
    content_tuple = tuple(f.read().splitlines())

print(f"\nContent read from file (as a tuple): {content_tuple}")
```

List all the Python features used in this program (e.g., print with separators, input, list, set, len, sorted, f-strings, conditional expressions, file writing, file reading, splitlines, tuple).

Answers

Section 1

1. `Hello, world!`
2. `#`
3. `print("Your Name")` (any name)

Section 2

1. Valid: b (`my_var`) and d (`_secret`). a starts with a digit, c contains a hyphen.
2. a) int, b) float, c) str, d) bool, e) NoneType
3. `<class 'int'>` and `<class 'str'>`
4. `age = 25; print(type(age))`
5. `bool(0) → False; bool("") → False`

Section 3

1. `One Two Three`
2. `print("Hello", "world", sep=", ")`
3. `print(f"{name} is {age} years old.")`
4. `=====` (10 pairs of `--`)
5. `print(1, 2, 3, sep="-")`

Section 4

1. a) 21, b) 36, c) 3, d) 3.333..., e) 16, f) 2
2. `w = 5.5; h = 3; area = w * h; print(area)`
3. `/` performs floating-point division; `//` performs integer division (floor division).

Section 5

1. 15
2. 3.14
3. (6, 2) (quotient and remainder)
4.

```
age_str = input("How old are you? ")
age = int(age_str)
print(f"Next year you will be {age + 1}.")
```
5. 11 (space counts as a character)

6. `min(numbers)`, `max(numbers)`, `sum(numbers)`
7. `<class 'builtin_function_or_method'>` (or similar)

Section 6

1. `colors = ["red", "green", "blue"]`
2. `colors[1]`
3. `"blue"`
4. `colors.append("yellow")`
5. `colors.insert(0, "purple")`
6. `colors.pop()`
7. Starting with 3, add 2 → 5, remove 1 → 4. So `len(colors)` is 4.
8. `nums[1:4]`
9. `[1, 2, 3, 4]` and `[1, 2, 1, 2, 1, 2]`
10. `"green" in colors`

Section 7

1. `t = (5, 10, 15)`
2. Tuples are immutable; lists are mutable.
3. No, because tuples are immutable.
4. `t.count(2) → 3; t.index(3) → 2`
5. `t = ("alone",)`
6. `(1, 2, 3, 4)`
7. Yes, e.g., `2 in (1,2,3) → True`

Section 8

1. `letters = {'a', 'b', 'c'}`
2. Duplicates are automatically removed: `{1, 2, 3}`
3. `letters.add('d')`
4. `letters.discard('b')` (or `remove` but `discard` is safer)
5.
 - o Union: `A | B` or `A.union(B)` → `{1,2,3,4,5,6}`
 - o Intersection: `A & B` or `A.intersection(B)` → `{3,4}`
 - o Difference A-B: `A - B` or `A.difference(B)` → `{1,2}`
 - o Symmetric difference: `A ^ B` or `A.symmetric_difference(B)` → `{1,2,5,6}`
6. `2 in A`
7. 3 (only unique elements count)
8. No, sets are unordered.

Section 9

1. `['Python', 'programming', 'is', 'fun']`
2. `" ".join(words)`
3. `"hello".upper()`
4. `"\n text \t ".strip()`
5. `"I love cats".replace("cats", "dogs")`
6. `"Python programming".find("pro")` → 7; if not found, returns -1.
7. `"photo.jpg".endswith(".jpg")` → True
8. `"apple pie".count('p')` → 2

Section 10

1.

```
with open("test.txt", "w") as f:  
    f.write("Hello, file!")
```
2.

```
with open("test.txt", "r") as f: content = f.read()
```
3. A list of strings, each ending with a newline character.
4.

```
with open("test.txt", "a") as f: f.write("\nAnother line")
```
5.

```
with open("data.txt", "w") as f:  
    f.write("\n".join(lines))
```
6. Use `read().splitlines()` or `read().split('\n')` but `splitlines` handles different line endings better.

Section 11

Features used:

- `print()` with string repetition (`"==" * 20`)
- `input()` for user input
- List creation `[word1, word2, ...]`
- `set()` to remove duplicates
- `list()` to convert back to list
- `len()` to get lengths
- f-strings for formatted output
- `sorted()` built-in function
- Conditional expressions (inline if) for printing only if index exists
- `with open()` for file writing
- `"\n".join()` to write list as lines
- `with open()` for file reading
- `read().splitlines()` to read lines without newlines

- `tuple()` to convert list to tuple