1. Try Block

A **try block** is used for **exception handling** — catching and managing **runtime errors**.

Purpose

- To **test** code that might cause an error.
- To **handle** the error gracefully if it occurs.
- To **prevent** your program from crashing.

Example

```
try:
    x = int(input("Enter a number: "))
    result = 10 / x
    print(result)
except ZeroDivisionError:
    print("You can't divide by zero!")
except ValueError:
    print("That's not a valid number.")
```

How it works:

- Python **tries** the code inside try:.
- If an **invalid instruction** occurs (e.g., dividing by zero), Python **jumps** to the matching except block.
- Execution **continues** after the try-except structure.

---

2. If Block

An **if block** is used for **conditional branching** — deciding **which code path** to execute.

Purpose

- To **check a condition**.
- To **branch execution** depending on whether the condition is True or False.

Example

```
x = int(input("Enter a number: "))

if x > 0:
    print("Positive number")
elif x < 0:
    print("Negative number")
else:
    print("Zero")
```

How it works:

- The condition after if is **evaluated**.
- If it's True, that block runs.
- If it's False, Python checks the next elif or else.

Key Difference

Feature	try block	if block
Purpose	Handle **unexpected runtime errors**	Control **program flow** based on conditions
Triggers	When something **goes wrong**	When a **condition is true**
Syntax	try: … except:	if: … elif: … else:
Used for	Crashes, invalid input, network issues, missing files, etc.	Checking values, ranges, states, etc.
Flow	Jumps to except **only on error**	Always **evaluates conditions**

Here's a cheat sheet of common exceptions:

Exception Name	When It Happens / Description	Example
*ValueError	Function receives the right type but invalid value	int("abc")
*TypeError	Operation or function applied to an invalid type	"hello" + 5
*ZeroDivisionError	Division or modulo by zero	10 / 0
*IndexError	Accessing a list/tuple with an invalid index	[1,2,3][5]
*KeyError	Accessing a dictionary with a non-existent key	{"a":1}["b"]
*FileNotFoundError	Opening a file that doesn't exist	open("nofile.txt")
IOError	General input/output problems	open("/protected/file.txt")
AttributeError	Accessing a non-existent attribute of an object	"abc".fake_method()
ImportError	Fails to import a module	import non_existent_module
*Exception	Base class for most built-in exceptions; can catch many errors	

```
try:
    1/0
except Exception:
    print("Error")
```

* EXCEPTIONS ARE MOST COMMON IN AN INTRO COURSE.