1. What is a Runnable?

Runnable is the **core abstraction** in modern LangChain.

Definition
A **Runnable** is a *building-block unit of work* that:

- takes an input
- transforms it into an output
- can be composed with other runnables
- supports batching, async, streaming, tracing, retries, etc.

A Runnable implements a **standard interface** with methods like:

- `invoke()` — run once
- `.batch()` / `.abatch()` — run in parallel
- `.stream()` / `.astream()` — stream output as it's generated
- chaining support using the pipe operator (`|`)([LangChain][1])

Examples of Runnables include:

- Prompt templates
- LLM / chat model wrappers
- Output parsers
- Retrievers
- Custom logic wrapped as a `RunnableLambda` ([Medium][2])

Key point:
Each of these implements the *Runnable interface*, so they can be combined seamlessly. ([LangChain][1])

---

2. What is LCEL (LangChain Expression Language)?

LCEL = LangChain Expression Language

Definition
LCEL is a *declarative syntax* for composing Runnables into chains (pipelines) using Python operators (e.g., `|`). It replaces the older `langchain.chains` API.

Instead of:

```
chain = LLMChain(...)
result = chain.run(...)
```

You now compose with a clear pipeline:

```
result = (prompt | llm | parser).invoke({"input": ...})
```

This works because `prompt`, `llm`, and `parser` are all **Runnables**. ([LangChain][3])

Benefits of LCEL

- Cleaner, declarative style
- Automatic support for sync/async/batching/streaming
- Parallel execution with `RunnableParallel`
- Better readability and modularity ([GeeksforGeeks][4])

> LCEL lets you *describe the pipeline* instead of writing imperative glue code. ([LangChain][3])

---

3. Why `langchain.chains` is gone

In LangChain **v1.x+**, the developers decided:

- Chains should be built *compositionally* using Runnables + LCEL
- The old `langchain.chains` structure was redundant and harder to optimize
- So it's no longer the primary API surface

Result:
Imports like:

```
from langchain.chains.combine_documents import ...
```

fail because those modules aren't part of the new design.

Instead, you **compose Runnables** using LCEL operators. ([LangChain][3])

---

4. How LCEL & Runnables Replace Chains

Using Runnables and LCEL, you can build anything you used to build with chain modules.

Basic example (prompt → model → parser)

```
from langchain_core.prompts import ChatPromptTemplate
from langchain_openai import ChatOpenAI
from langchain_core.output_parsers import StrOutputParser

prompt = ChatPromptTemplate.from_template("Summarize: {text}")
model = ChatOpenAI(model="gpt-4o-mini")
parser = StrOutputParser()

pipeline = prompt | model | parser

output = pipeline.invoke({"text": "LCEL stands for LangChain Expression I
print(output)
```

Here:

- `prompt` is a Runnable
- `model` is a Runnable
- `parser` is a Runnable
- `|` composes them into a sequence ([LangChain][5])

---

5. Parallel & Complex Pipelines

You can also compose Runnables in parallel:

```
from langchain_core.runnables import RunnableParallel

parallel_stage = RunnableParallel({
    "a": run_a,
    "b": run_b
})

full_pipeline = parallel_stage | postprocess_runnable
```

This is the LCEL way to build workflows that used to require separate chain helpers. ([LangChain][1])

---

6. Simple Runnable Example

Here's an **illustrative simple pipeline**:

```
from langchain_core.runnables import RunnableLambda

# A simple runnable that adds 1
inc = RunnableLambda(lambda x: x + 1)

# Another that multiplies
mul2 = RunnableLambda(lambda x: x * 2)

# Compose with LCEL
pipeline = inc | mul2

print(pipeline.invoke(3))  # prints 8
```

Explanation:

1. `inc.invoke(3)` → 4
2. `mul2.invoke(4)` → 8
3. `pipeline.invoke(3)` does both in sequence ([LangChain][1])

---

7. Summary

Runnable

- The *core abstraction* in LangChain v1.x+
- Encapsulates a unit of work
- Standard interface with `invoke`, `batch`, `stream`, etc. ([LangChain][1])

LCEL (LangChain Expression Language)

- A *declarative way* to compose Runnables into pipelines
- Uses `|` and other operators to build sequences
- Replaces the old `chains` API ([LangChain][3])

No `langchain.chains`

- Old helper modules are removed or deprecated in v1.x+
- Pipelines now built with LCEL + Runnables ([LangChain][3])

Comparison (Old vs New)

Concept	Old API (pre v1)	New API (v1+)	
Chains	`langchain.chains.*`	Composed via LCEL (`	`)
Chain helpers	helpers like `LLMChain`	`Runnable` + LCEL	
Composition	internal chain classes	Declarative pipelines	
Parallelism	custom code	`RunnableParallel`	
Streaming	partial	`run.stream()` / `run.astream()`	([LangChain][6])