
- 출처: LangChain 공식 문서 또는 해당 교재명
- 원본 URL: https://smith.langchain.com/hub/teddynote/summary-stuff-documents

---

4. `LLM 체인 라우팅 (RunnableLambda, RunnableBranch)`

1) `RunnableBranch`

- 개념
  - 입력에 따라 동적으로 로직을 라우팅할 수 있는 강력한 도구
  - **`입력 데이터의 특성에 기반하여 다양한 처리 경로를 유연하게 정의 가능`**

- 장점
  - 복잡한 의사 결정 트리를 `간단`하고 `직관적인 방식`으로 구현
    - `코드`의 `가독성`과 `유지보수성`을 크게 `향상`시킴
    - `로직`의 `모듈화`와 `재사용성` `촉진`시킴
  - `런타임`에 `동적`으로 `분기 조건`을 `평가`하고 `적절한 처리 루틴`을 `선택` → 시스템의 `적응력`과 `확장성` 높임
  - `다양한 도메인`에서 활용 → 입력 데이터의 다양성과 변동성이 큰 애플리케이션 개발에 매우 유용
  - *코드의 복잡성을 줄이고, 시스템의 유연성과 성능을 향상시킬 수 있음*

- **`입력에 따른 동적 로직 라우팅`**
  - 동적 로직 = `LCEL`에서는 이전 단계의 출력이 다음 단계를 정의하는 *`비결정적`* 체인을 생성할 수 있음
  - 라우팅: `LLM`과의 상호작용 구조와 일관성을 제공하는데 도움이 됨
    - 라우팅 방법 a. **`RunnableLamba`**에서 **`조건부로 실행 가능한 객체 반환`** *(권장)*
    - 라우팅 방법 b. **`RunnableBranch`** 사용
      - 공통점: 모두 첫 번째 단계에서 입력 질문을 *`수학`*, *`과학`* 또는 *`기타`*에 대한 것으로 분류 → 해당 프롬프트 체인으로 라우팅

- **`환경설정`**

```
# API 키를 환경변수로 관리하기 위한 설정 파일
from dotenv import load_dotenv

# API 키 정보 로드
load_dotenv()                           # True
```

```
from langsmith import Client
from langsmith import traceable

import os

# LangSmith 환경 변수 확인

print("\n--- LangSmith 환경 변수 확인 ---")
langchain_tracing_v2 = os.getenv('LANGCHAIN_TRACING_V2')
langchain_project = os.getenv('LANGCHAIN_PROJECT')
langchain_api_key_status = "설정됨" if os.getenv('LANGCHAIN_API_KEY') else "설정되지 않음" # API 키 값은 직접 출력하지 않음

if langchain_tracing_v2 == "true" and os.getenv('LANGCHAIN_API_KEY') and langchain_project:
    print(f"✅ LangSmith 추적 활성화됨 (LANGCHAIN_TRACING_V2='{langchain_tracing_v2}')")
    print(f"✅ LangSmith 프로젝트: '{langchain_project}'")
    print(f"✅ LangSmith API Key: {langchain_api_key_status}")
    print("  -> 이제 LangSmith 대시보드에서 이 프로젝트를 확인해 보세요.")
else:
    print("❌ LangSmith 추적이 완전히 활성화되지 않았습니다. 다음을 확인하세요:")
    if langchain_tracing_v2 != "true":
        print(f"  - LANGCHAIN_TRACING_V2가 'true'로 설정되어 ✦  않습니다 (현재: '{langchain_tracing_v2}').")
```

```
    if not os.getenv('LANGCHAIN_API_KEY'):
        print("  - LANGCHAIN_API_KEY가 설정되어 있지 않습니다.")
    if not langchain_project:
        print("  - LANGCHAIN_PROJECT가 설정되어 있지 않습니다.")
```

- 셀 출력

```
--- LangSmith 환경 변수 확인 ---
✅ LangSmith 추적 활성화됨 (LANGCHAIN_TRACING_V2='true')
✅ LangSmith 프로젝트: 'LangChain-prantice'
✅ LangSmith API Key: 설정됨
-> 이제 LangSmith 대시보드에서 이 프로젝트를 확인해 보세요.
```

- **간단한 예시**
  - **공통 분류 체인 생성**: 들어오는 질문 → *수학*, *과학*, *기타* 중 하나로 분류하는 `Chain` 생성

```
from langchain_google_genai import ChatGoogleGenerativeAI
from langchain_core.output_parsers import StrOutputParser
from langchain_core.prompts import ChatPromptTemplate
from dotenv import load_dotenv
import os                                       # 7.8s
```

```
# API 키 확인
if not os.getenv("GOOGLE_API_KEY"):
    os.environ["GOOGLE_API_KEY"] = input("Enter your Google API key: ")

# LLM 초기화
gemini_lc = ChatGoogleGenerativeAI(
    model="gemini-2.5-flash-lite",
)
```

- `LLM` 생성 = `gemini-2.5-flash-lite`

```
E0000 00:00:1759975736.788629 1692296 alts_credentials.cc:93] ALTS creds ignored. Not running on GCP and untruste
```

```
# 템플릿 생성하기
from langchain_core.prompts import PromptTemplate

prompt = PromptTemplate.from_template(
    """주어진 사용자 질문을 `수학`, `과학`, 또는 `기타` 중 하나로 분류하세요. 한 단어 이상으로 응답하지 마세요.

<question>
{question}
</question>

Classification:"""
)
```

```
# 체인 생성하기

chain = (
    prompt
    | gemini_lc
    | StrOutputParser()         # 문자열 출력 파서 사용하기
)
```

- 생성한 `Chain`을 사용해 질문 분류하기

```
# query_1
# 질문 입력 → 체인 호출하기
chain.invoke({"question": "2+2 는 무엇인가요?"})
```

- `query_1` (`"question": "2+2 는 무엇인가요?"`) - (`1.0s`)
  - '수학'

```
# query_2
# 질문 입력 → 체인 호출하기
chain.invoke({"question": "작용 반작용의 법칙은 무엇인가요?"})
```

- `query_2` (`"question": "작용 반작용의 법칙은 무엇인가요?"`) - (`0.7s`)
  - '과학'

```
# query_3
# 질문 입력 → 체인 호출하기
chain.invoke({"question": "Google은 어떤 회사인가요?"})
```

- `query_3` (`"question": "Google은 어떤 회사인가요?"`) - (`0.6s`)
  - '기타'

- 3개의 하위 체인 생성하기

```
# 수학 설명 체인
math_chain = (
    PromptTemplate.from_template(
        """You are an expert in math. \
Always answer questions starting with "깨봉선생님께서 말씀하시기를..". \
Respond to the following question:

Question: {question}
Answer:"""
    )
    | gemini_lc
)

# 과학 설명 체인
science_chain = (
    PromptTemplate.from_template(
        """You are an expert in science. \
Always answer questions starting with "아이작 뉴턴 선생님께서 말씀하시기를..". \
Respond to the following question:

Question: {question}
Answer:"""
    )
    | gemini_lc
)

# 기타 설명 체인
general_chain = (
    PromptTemplate.from_template(
        """Respond to the following question concisely:

Question: {question}
Answer:"""
    )
    | gemini_lc
)
```

- **`사용자 정의 함수 사용하기`**
  - `LangChain` 공식 도큐먼트에서 권장하는 방식
  - 서로 다른 출력 간의 라우팅을 위해 **`사용자 정의 함수`**를 **`RunnableLambda`**로 래핑 → **활용 가능**

```
def route(info):
    # 주제에 "수학"이 포함되어 있는 경우
    if "수학" in info["topic"].lower():
        # datascience_chain을 반환
        return math_chain

    # 주제에 "과학"이 포함되어 있는 경우
    elif "과학" in info["topic"].lower():
        # art_chain을 반환

        return science_chain
    # 그 외의 경우
    else:
```

```
        # general_chain을 반환
        return general_chain
```

```
from operator import itemgetter
from langchain_core.runnables import RunnableLambda

full_chain = (
    {"topic": chain, "question": itemgetter("question")}
    | RunnableLambda(
        route               # 경로를 지정하는 함수를 인자로 전달함
    )
    | StrOutputParser()
)
```

```
# full_chain_query_1
# 수학과 관련된 질문 입력 → 체인 호출하기
full_chain.invoke({"question": "미적분의 개념에 대해 말씀해 주세요."})
```

- `full_chain_query_1` (`"question": "미적분의 개념에 대해 말씀해 주세요."`) - (`3.9s`)

```
'깨봉선생님께서 말씀하시기를..

미적분은 크게 두 가지 중요한 개념으로 나눌 수 있습니다. 바로 **미분**과 **적분**입니다. 이 두 가지는 서로 역연산 관계에 있으며, 함수의 변화율과 

**1. 미분 (Differentiation): 변화율의 탐구**

미분은 **순간적인 변화율**을 구하는 과정입니다. 마치 자동차의 속도계가 특정 순간의 속도를 보여주듯이, 미분은 함수가 특정 지점에서 얼마나 빠르게 변하

* **핵심 아이디어:** 아주 작은 두 점 사이의 평균 변화율을 구하고, 그 두 점 사이의 간격을 0에 가깝게 줄였을 때의 극한값을 구하는 것입니다.
* **기하학적 의미:** 함수의 그래프에서 특정 점에서의 **접선의 기울기**를 의미합니다. 즉, 그 점에서 함수가 얼마나 가파르게 올라가거나 내려가는지를
* **활용:**
  * 속도, 가속도 계산 (물리학)
  * 최댓값, 최솟값 찾기 (최적화 문제)
  * 함수의 증가/감소 구간 파악
  * 곡선의 모양 분석

**2. 적분 (Integration): 누적된 양의 계산**

적분은 **변화하는 양을 누적**하여 총량을 구하는 과정입니다. 마치 자동차가 일정 시간 동안 이동한 총 거리를 계산하듯이, 적분은 함수의 그래프 아래 넓이

* **핵심 아이디어:** 아주 작은 구간에서의 함숫값을 곱한 직사각형들의 넓이를 모두 더하고, 그 구간의 간격을 0에 가깝게 줄였을 때의 극한값을 구하는 
* **기하학적 의미:** 함수의 그래프와 x축 사이의 **넓이**를 의미합니다.
* **활용:**
  * 넓이, 부피, 길이 계산
  * 질량, 중심, 관성 모멘트 계산 (물리학, 공학)
  * 확률 계산 (확률 및 통계)
  * 미분 방정식의 해를 구하는 데 사용 (미분과 적분의 역연산 관계)

**미분과 적분의 관계 (미적분의 기본 정리)**

미분과 적분은 서로를 되돌리는 관계에 있습니다. 이것을 **미적분의 기본 정리**라고 하며, 미적분학에서 가장 중요하고 아름다운 정리 중 하나입니다.

* 어떤 함수를 미분한 결과(도함수)를 다시 적분하면 원래 함수로 돌아옵니다.
* 어떤 함수를 적분한 결과(부정적분)를 미분하면 원래 함수가 됩니다.

이러한 미적분학은 자연 현상을 이해하고, 다양한 문제를 해결하는 데 필수적인 수학적 도구입니다.'
```

```
# full_chain_query_2
# 과학과 관련된 질문 입력 → 체인 호출하기
full_chain.invoke({"question": "중력은 어떻게 작용하나요?"})
```

- `full_chain_query_2` (`"question": "중력은 어떻게 작용하나요?"`) - (`2.3s`)

```
'아이작 뉴턴 선생님께서 말씀하시기를..

중력은 질량을 가진 모든 물체가 서로 끌어당기는 힘입니다. 이 힘의 크기는 두 물체의 질량에 비례하고, 두 물체 사이의 거리의 제곱에 반비례합니다. 즉, 질

좀 더 자세히 설명하자면, 뉴턴 선생님께서는 만유인력의 법칙을 통해 중력을 설명하셨습니다. 이 법칙은 다음과 같은 수식으로 표현될 수 있습니다.

$F = G \\frac{m_1 m_2}{r^2}$
```

```
여기서:
  * $F$는 두 물체 사이에 작용하는 중력의 크기입니다.
  * $G$는 만유인력 상수라고 불리는 값으로, 중력의 보편적인 세기를 나타냅니다.
  * $m_1$과 $m_2$는 각각 두 물체의 질량입니다.
  * $r$은 두 물체 중심 사이의 거리입니다.

이 법칙에 따르면, 만약 두 물체의 질량이 두 배가 되면 중력도 두 배가 됩니다. 하지만 두 물체 사이의 거리가 두 배가 되면 중력은 네 배 약해집니다 (거

이 중력 때문에 지구는 우리를 끌어당기고, 달은 지구 주위를 돌며, 태양계의 행성들이 태양 주위를 공전하는 것입니다. 또한, 이 힘은 눈에 보이지 않지만 우
```

```
# full_chain_query_3
# 기타 질문 입력 → 체인 호출하기
full_chain.invoke({"question": "RAG(Retrieval Augmented Generation)은 무엇인가요?"})
```

- `full_chain_query_3` (`"question": "RAG(Retrieval Augmented Generation)은 무엇인가요?"`) - (`1.3s`)

```
'RAG(Retrieval Augmented Generation)는 외부 지식 소스를 검색하여 답변을 생성하는 AI 모델입니다.'
```

---

2) `RunnableBranch`

- **`RunnableBranch`**
  - 입력값에 따라 실행할 조건, `Runnable`을 정의할 수 잇는 특별한 유형의 `Runnable`
  - *위에서 설명한 사용자 정의 함수로 구현할 수 없는 기능을 제공하지 않음* → **`사용자 정의 함수 사용하는 것이 좋음`**

- 문법
  - **`RunnableBranch`** = **(** `조건`, **`Runnable`** **) 쌍의 리스트**, 기본 `Runnable`로 초기화
  - 호출 ⟶ 전달된 입력값을 각 조건에 전달 ⟶ 분기 선택
    - `True`로 평가되는 `첫 번째 조건` 선택 ⟶ 해당 조건에 해당하는 `Runnable`을 입력값과 함께 실행
    - 제공된 조건과 일치하는 것이 없으면 **`기본 Runnable`** **실행**

```
from operator import itemgetter
from langchain_core.runnables import RunnableBranch

branch = RunnableBranch(
    # 주제에 "수학"이 포함되어 있는지 확인 → 포함되어 있다면 math_chain 실행
    (lambda x: "수학" in x["topic"].lower(), math_chain),

    # 주제에 "과학"이 포함되어 있는지 확인 → 포함되어 있다면 science_chain 실행
    (lambda x: "과학" in x["topic"].lower(), science_chain),

    # 위의 조건에 해당하지 않는 경우 general_chain 실행
    general_chain,
)

# 주제와 질문을 입력받음 → branch를 실행하는 전체 체인 정의함
full_chain = (
    {"topic": chain, "question": itemgetter("question")}
    | branch
    | StrOutputParser()
)
```

```
# brach_query_1
# 질문 입력 → 전체 체인 실행하기
full_chain.invoke({"question": "미적분의 개념에 대해 말씀해 주세요."})
```

- `brach_query_1` (`"question": "미적분의 개념에 대해 말씀해 주세요."`) - (`4.8s`)

```
"깨봉선생님께서 말씀하시기를..

미적분은 크게 두 가지 주요 개념으로 나눌 수 있습니다. 바로 **미분(Differentiation)**과 **적분(Integration)**입니다. 이 둘은 서로 역연산
```

1. 미분 (Differentiation): 변화율의 탐구

미분은 어떤 양이 다른 양에 따라 **얼마나 빠르게 변하는지**를 나타내는 개념입니다. 즉, **순간적인 변화율**을 구하는 것입니다.

* **직관적인 이해:**
* **함수의 기울기:** 가장 기본적인 예시는 함수의 그래프에서 특정 점에서의 **순간적인 기울기**를 구하는 것입니다. 마치 자동차의 속도계가 현재 속도
* **접선:** 함수의 그래프에 특정 점에서 그은 **접선의 기울기**가 바로 그 점에서의 미분값입니다.
* **핵심 개념:**
* **극한 (Limit):** 미분은 극한의 개념을 기반으로 합니다. 두 점 사이의 평균 변화율을 구한 후, 두 점 사이의 간격을 0으로 극한을 취함으로써 순간
* **도함수 (Derivative):** 원래 함수 $f(x)$로부터 그 함수의 각 점에서의 변화율을 나타내는 새로운 함수 $f'(x)$를 얻는 과정을 미분이라고 하
* **응용:**
* 물리학: 속도, 가속도, 힘 등을 계산합니다.
* 경제학: 한계 비용, 한계 수입 등을 분석합니다.
* 공학: 최적화 문제, 시스템의 동적 변화 분석 등에 사용됩니다.

2. 적분 (Integration): 누적의 탐구

적분은 미분과는 반대로, **변화하는 양을 모두 더하거나 쌓아서 전체적인 양을 구하는** 개념입니다. 즉, **누적된 값**을 구하는 것입니다.

* **직관적인 이해:**
* **넓이:** 곡선으로 둘러싸인 영역의 **넓이**를 구하는 것이 적분의 대표적인 예입니다. 작은 직사각형들을 무수히 많이 더해서 전체 넓이를 구하는 것으
* **부피:** 입체의 부피를 구하는 데도 사용됩니다.
* **핵심 개념:**
* **부정적분 (Indefinite Integral):** 어떤 함수를 미분했을 때 원래 함수가 되는 함수를 찾는 과정입니다. 이는 미분의 역연산에 해당하며, 적분
* **정적분 (Definite Integral):** 특정 구간 $[a, b]$에서 함수의 그래프 아래의 넓이, 즉 누적된 값을 구하는 것입니다. 부정적분을 이용하여
* **응용:**
* 물리학: 이동 거리, 일의 양, 압력 등을 계산합니다.
* 확률 및 통계: 확률 밀도 함수의 넓이를 통해 누적 확률을 계산합니다.
* 공학: 누적된 에너지, 질량 분포 등을 계산합니다.

미적분의 기본 정리 (Fundamental Theorem of Calculus)

이 두 개념, 미분과 적분은 **미적분의 기본 정리**에 의해 밀접하게 연결됩니다. 이 정리는 미분과 적분이 서로 역연산 관계에 있음을 수학적으로 증명하며,

요약하자면, 미적분은 **변화하는 세계를 이해하고 분석하는 데 필수적인 수학 언어**라고 할 수 있습니다. 순간의 변화를 파악하는 미분과, 누적된 결과를 ㅈ

```
# brach_query_2
# 질문 입력 → 전체 체인 실행하기
full_chain.invoke({"question": "중력 가속도는 어떻게 계산하나요?"})
```

- `brach_query_2` (`"question": "중력 가속도는 어떻게 계산하나요?"`) - (`3.6s`)

"아이작 뉴턴 선생님께서 말씀하시기를.. 중력 가속도는 물체와 지구 사이의 만유인력에 의해 결정됩니다. 이 중력 가속도를 계산하는 가장 기본적인 방법은

1. 뉴턴의 만유인력 법칙 이용:

뉴턴 선생님께서 발견하신 만유인력 법칙에 따르면, 두 질량체 사이에 작용하는 만유인력(F)은 다음과 같습니다.

$F = G \\frac{m_1 m_2}{r^2}$

여기서:
  * F는 만유인력의 크기입니다.
  * G는 만유인력 상수 (약 $6.674 \\times 10^{-11} \\text{ N m}^2/\\text{kg}^2$) 입니다.
  * m_1은 지구의 질량입니다.
  * m_2는 물체의 질량입니다.
  * r은 지구 중심으로부터 물체까지의 거리 (지구의 반지름 + 물체의 높이) 입니다.

2. 뉴턴의 제2 운동 법칙 이용:

한편, 뉴턴 선생님께서 말씀하신 제2 운동 법칙에 따르면, 물체에 작용하는 힘(F)은 물체의 질량(m_2)과 가속도(a)의 곱과 같습니다.

$F = m_2 a$\n\n중력 가속도는 바로 이 힘(F)에 의해 발생하는 가속도(a)를 의미하므로, 위 두 식을 결합하면 다음과 같이 됩니다.

$m_2 a = G \\frac{m_1 m_2}{r^2}$

양변에서 물체의 질량(m_2)을 소거하면, 중력 가속도(a)를 계산하는 식이 나옵니다.

$a = G \\frac{m_1}{r^2}$

```
**이것이 바로 중력 가속도를 계산하는 기본적인 원리입니다.**

**실제 계산:**

  * **지구의 질량 ($m_1$)**: 약 $5.972 \\times 10^{24} \\text{ kg}$
  * **지구의 평균 반지름 ($r$)**: 약 $6.371 \\times 10^6 \\text{ m}$

이 값들을 위 식에 대입하면, 지구 표면에서의 중력 가속도는 약 $9.8 \\text{ m/s}^2$ 가 됩니다.

**주의할 점:**

  * **고도에 따른 변화:** 지구 표면에서 위로 올라갈수록 $r$ 값이 커지므로 중력 가속도는 작아집니다.
  * **지구의 모양:** 지구는 완벽한 구가 아니므로, 지역에 따라 중력 가속도가 미세하게 다를 수 있습니다.
  * **지구 자전의 영향:** 지구의 자전으로 인해 원심력이 발생하며, 이 또한 중력 가속도에 약간의 영향을 미칩니다.

따라서, 일반적으로 '중력 가속도'라고 할 때는 지구 표면에서의 평균값을 의미하며, 이는 약 $9.8 \\text{ m/s}^2$ 로 사용됩니다."
```

```
# brach_query_3
# 질문 입력 → 전체 체인 실행하기
full_chain.invoke({"question": "RAG(Retrieval Augmented Generation)은 무엇인가요?"})
```

- `brach_query_3` (`"question": "RAG(Retrieval Augmented Generation)은 무엇인가요?"`) - (`1.5s`)

```
'RAG는 **외부 지식 데이터베이스에서 관련 정보를 검색하여 언어 모델의 답변 생성에 활용하는 기술**입니다.'
```

- next: ***`05. RunnableParallel`***
