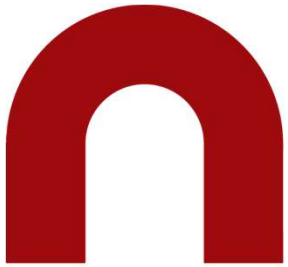


Instituto Superior de Engenharia

Politécnico de Coimbra

Integração de Dados

CTeSP Tecnologias e Programação de Sistemas de Informação (Cantanhede)

Professor: João Leal

joao.leal@isec.pt

UNIgreen The green European University

PRR Plano de Recuperação e Resiliência

REPÚBLICA PORTUGUESA

Financiado pela União Europeia NextGenerationEU

Introdução ao ETL

- **ETL** é a sigla para *Extract, Transform, Load* (Extração, Transformação e Carga).

- Trata-se de um **processo de integração de dad**os que consiste em **recolher dados de múltiplas fontes, convertê-los num formato consistente e útil, e carregá-los para um repositório centralizado**, como um *Data Warehouse* ou *Data Lake*.

Introdução ao ETL

- O objetivo principal do ETL é **consolidar dados de sistemas** díspares para criar uma ***"fonte única da verdade"*** (*single source of truth*), permitindo análises de dados, relatórios de *Business Intelligence (BI)* e a **tomada de decisões estratégicas mais informadas.**

Objetivo: garantir que os dados utilizados em relatórios e análises estão consolidados, limpos e prontos para uso.

Introdução ao ETL

- O conceito de ETL surgiu na década de 1970, impulsionado pela necessidade das organizações de integrar dados de diferentes bases/bancos de dados e sistemas de informação que começavam a proliferar.

- Com o aparecimento dos primeiros *Data Warehouses* no final dos anos 80 e início dos 90, o ETL tornou-se o método padrão para alimentar estes repositórios com dados consolidados e prontos para análise.

Introdução ao ETL

- Inicialmente, os processos ETL *eram desenhados para lidar com dados estruturados de bases de dados relacionais*.

- Com a explosão do Big Data, **o ETL evoluiu para conseguir processar grandes volumes de dados não estruturados e semiestruturados**, provenientes de fontes como redes sociais, dispositivos IoT e APIs web.

Introdução ao ETL

- Esta evolução também deu origem a abordagens alternativas, como o ***ELT (Extract, Load, Transform)***, que se adapta melhor às **arquiteturas de dados modernas baseadas na nuvem.**

ETL no Contexto Empresarial

- Visão Consolidada do Negócio: Permite que as empresas combinem dados de diferentes departamentos (vendas, marketing, finanças, etc.) e sistemas (CRM, ERP, etc.) para obter uma visão completa e unificada das suas operações.

- Melhora da Qualidade e Consistência dos Dados: A fase de transformação é crucial para limpar, padronizar e validar os dados, garantindo que a informação utilizada para análise é precisa, consistente e fiável.

- Base para Business Intelligence e Análise de Dados: Fornece os dados preparados e estruturados necessários para ferramentas de BI, relatórios e dashboards. Sem um processo ETL eficaz, a análise de dados seria mais lenta, mais propensa a erros e menos fiável.

- Eficiência Operacional: Automatiza a tarefa, muitas vezes complexa e

ETL no Contexto Empresarial

- Eficiência Operacional: Automatiza a tarefa, muitas vezes complexa e demorada, de recolher e preparar dados, libertando os analistas e engenheiros de dados para se concentrarem em tarefas de maior valor, como a criação de modelos e a extração de insights.

- Contexto Histórico: Ao integrar dados históricos com informações atuais, o ETL permite a análise de tendências e padrões ao longo do tempo, algo essencial para previsões e planeamento estratégico.

ETL em Data Warehousing e Business Intelligence

Instituto Superior de Engenharia
Politécnico de Coimbra

- ***Data Warehousing:*** armazena dados históricos de várias fontes (ERP, CRM, vendas, produção).

- **ETL:** prepara os dados para o *Data Warehouse*, permitindo comparações e análises consistentes.

- ***Business Intelligence (BI):*** ferramentas como *Power BI* ou *Tableau* usam os dados carregados pelo ETL para gerar relatórios e *dashboards*.

ETL em Data Warehousing e Business Intelligence

Instituto Superior de Engenharia
Politécnico de Coimbra

- Sem ETL, os dados estariam dispersos, incompletos e inconsistentes.

Arquitetura geral de um pipeline ETL

Instituto Superior de Engenharia
Politécnico de Coimbra

- ***Extract:*** dados vindos de várias fontes (ficheiros, BD, APIs).
- ***Transform:*** normalização, limpeza, enriquecimento, agregação.
- ***Load:*** carregamento em DW, DL ou outra BD.

[Fontes] → [Extract] → [Transform] → [Load] → [Data Warehouse] → [BI Reports]

Fase 1: Extração (Extract)

- A primeira fase consiste em **recolher ou extrair os dados das suas fontes de origem**. Estas fontes podem ser extremamente variadas e heterogéneas.
- **Fontes de Dados Comuns**:
  - Bancos de dados relacionais (SQL Server, Oracle, MySQL).
  - Bancos de dados NoSQL (MongoDB, Cassandra).
  - Sistemas empresariais como ERPs e CRMs.
  - Ficheiros planos (CSV, JSON, XML).
  - APIs de serviços web.
  - Sensores de dispositivos IoT.

Fase 1: Extração (Extract)

- Os **dados extraídos são geralmente movidos para uma área de preparação** (*staging area*), que é um repositório intermédio onde os dados são armazenados temporariamente antes de serem transformados.

Fase 1: Extração (Extract)

Métodos de Extração:

- ***Carga Completa (Full Load):*** Todos os dados da fonte são extraídos de uma só vez. Este método é mais simples, mas pode ser impraticável para grandes volumes de dados.

- ***Carga Incremental (Incremental Load):*** Apenas os dados novos ou modificados desde a última extração são recolhidos. Este método é mais eficiente, pois reduz o volume de dados transferidos e o tempo de processamento.

Fase 1: Extração (Extract)

Resumindo:

- **Full Extraction:** extrair todos os dados de uma fonte.

  Mais simples, mas pesado.

- **Incremental Extraction:** só extrai novos registos ou registos alterados.

  Mais eficiente, usado em produção.

- **Change Data Capture (CDC):** monitoriza alterações em tempo real (inserções, *updates*, deletes).

Fase 1: Extração (Extract)


```python
import sqlite3
import pandas as pd

# Conectar a uma base SQLite
conn = sqlite3.connect("base_dados.db")

# Ler dados da tabela clientes
clientes = pd.read_sql("SELECT * FROM clientes", conn)

print(clientes.head())
conn.close()
```

Fase 2: Transformação (Transform)

- Esta é a **fase mais complexa** e onde **o maior valor é acrescentado**.

- Os **dados brutos extraídos são processados e convertidos num formato limpo, consistente e adequado para o sistema de destino**.

Fase 2: Transformação (Transform)

Instituto Superior de Engenharia
Politécnico de Coimbra

Operações Comuns de Transformação:

- ***Limpeza (Cleansing):*** Corrigir ou remover dados incorretos, inconsistentes ou duplicados e tratar valores em falta (nulos).

- ***Padronização (Standardization):*** Assegurar que os dados seguem um formato uniforme. Por exemplo, converter todas as datas para o formato AAAA-MM-DD ou unidades de medida para um padrão comum.

Fase 2: Transformação (Transform)

Instituto Superior de Engenharia
Politécnico de Coimbra

- ***Agregação (Aggregation):*** Resumir dados para um nível de granularidade mais alto, como calcular o total de vendas diárias a partir de transações individuais.

- ***Junção (Joining):*** Combinar dados de diferentes fontes. Por exemplo, juntar dados de clientes de um CRM com dados de vendas de um sistema transacional.

Fase 2: Transformação (Transform)

- ***Validação (Validation):*** Aplicar regras para garantir a integridade e a qualidade dos dados (ex: verificar se um código postal tem o formato correto).

- ***Enriquecimento (Enrichment):*** Adicionar novos dados calculados ou derivados dos dados existentes, como calcular a idade de um cliente a partir da sua data de nascimento.

Fase 2: Transformação (Transform)

Instituto Superior de Engenharia
Politécnico de Coimbra

```python
df = pd.DataFrame({
    "id": [1, 2, 3, 4],
    "nome": ["ana", "PEDRO", "Maria", "joao"],
    "data": ["2025-09-01", "2025/09/02", "01-09-2025", "2025-09-03"],
    "valor": [10.5, None, 20.0, 15.0]
})

# Normalizar nomes
df["nome"] = df["nome"].str.upper()

# Normalizar datas
df["data"] = pd.to_datetime(df["data"], errors="coerce")

# Tratar valores nulos
df["valor"].fillna(0, inplace=True)

# Calcular total acumulado
df["total_acumulado"] = df["valor"].cumsum()

print(df)
```

Fase 3: Carga (Load)

- A fase final consiste em **carregar os dados transformados para o sistema de destino final**.

- **Sistemas de Destino Comuns:**
  - ***Data Warehouse (DW):*** Um repositório central de dados estruturados, otimizado para consulta e análise.
  - ***Data Lake:*** Um repositório que armazena grandes volumes de dados brutos, tanto estruturados como não estruturados.

Fase 3: Carga (Load)

Métodos de Carga:

- ***Carga Completa (Full Load):*** Também conhecida como "destrutiva", **apaga todos os dados existentes na tabela de destino e insere o novo conjunto de dados**. É usada para tabelas pequenas ou quando os dados históricos não precisam de ser mantidos.

- ***Carga Incremental (Incremental Load):*** **Adiciona novos registos** (*append*) **ou atualiza registos existentes** (*update/insert*) no destino. Este método é mais comum em *Data Warehouses* para **preservar o histórico e otimizar o desempenho**.

Fase 3: Carga (Load)


```python
import sqlite3

# Carregar dados transformados para SQLite
conn = sqlite3.connect("datawarehouse.db")

df.to_sql("tabela_final", conn, if_exists="replace", index=False)

print("Carga concluída com sucesso!")
conn.close()
```

ETL vs. ELT

- À medida que as arquiteturas de dados evoluíram, especialmente com o **Big Data** e das plataformas de computação em nuvem, surgiu uma nova abordagem à integração de dados: o ***ELT (Extract, Load, Transform).***

- Compreender as diferenças entre ETL e ELT é fundamental para escolher a arquitetura correta para um determinado projeto.

ETL vs. ELT

- O ELT é um processo de integração de dados que **inverte as duas últimas fases do ETL tradicional**.

ETL vs. ELT

A sequência de operações é a seguinte:

- **Extract (Extração):** Os dados são extraídos das fontes de origem, de forma semelhante ao ETL.
- **Load (Carga):** **Em vez de serem enviados para uma área de preparação para transformação, os dados brutos são imediatamente carregados diretamente no sistema de destino**, que é tipicamente um *Data Warehouse* ou *Data Lake* na nuvem (como *Google BigQuery, Amazon Redshift ou Snowflake*).
- **Transform (Transformação):** A **transformação dos dados ocorre dentro do sistema de destino**. As regras de negócio, limpeza e padronização são aplicadas aos dados já carregados, aproveitando o poder de processamento massivo e paralelo do *Data Warehouse* moderno.

ETL vs. ELT

- A escolha entre ETL e ELT depende de vários fatores, como o volume de dados, a infraestrutura disponível e os objetivos de análise.

ETL vs. ELT

Devemos utilizar ETL quando:

- Os dados são maioritariamente **estruturados e provêm de fontes relacionais**.

- As transformações são **complexas, pesadas e requerem um motor dedicado**.

- Existem **fortes requisitos de segurança** e conformidade que exigem que os **dados sensíveis sejam limpos ou mascarados antes de entrarem no *Data Warehouse***.

ETL vs. ELT

- A infraestrutura de destino (*Data Warehouse*) **não tem poder de processamento suficiente para realizar as transformações** de forma eficiente.

- O **volume de dados não é massivo** (embora o ETL também possa lidar com grandes volumes, o ELT é **geralmente mais escalável para Big Data**).

ETL vs. ELT

Devemos utilizar ELT quando:

- Lidamos com **grandes volumes de dados** (Big Data), incluindo dados não estruturados ou semiestruturados.

- A **velocidade de ingestão dos dados é uma prioridade**. O ELT permite que os **dados estejam disponíveis para consulta quase imediatamente após a extração.**

Ferramentas e Tecnologias de ETL

- Existem inúmeras ferramentas e tecnologias disponíveis para implementar processos de ETL, desde soluções comerciais robustas a ferramentas open-source e bibliotecas de programação.

- A escolha depende do orçamento, da complexidade do projeto, da infraestrutura existente e das competências da equipa.

Ferramentas ETL com Interface Visual (GUI)

- Estas ferramentas permitem **criar fluxos de ETL** (chamados de *jobs ou transformations*) através de uma **interface gráfica** de "arrastar e soltar" (*drag-and-drop*), o que **acelera o desenvolvimento e facilita a manutenção**.

Ferramentas ETL com Interface Visual (GUI)

Pentaho Data Integration (PDI), também conhecido como Kettle:

- **Descrição:** Uma das ferramentas ETL open-source mais populares e poderosas. Permite desenhar fluxos de dados complexos que conectam a uma vasta gama de fontes e destinos.
- **Vantagens:** Gratuito (na versão community), muito flexível, grande comunidade de utilizadores e uma curva de aprendizagem relativamente suave para tarefas comuns.
- **Ideal para:** Empresas que procuram uma solução robusta sem o custo de uma licença comercial.

Ferramentas ETL com Interface Visual (GUI)

Talend Open Studio for Data Integration:

- **Descrição:** Concorrente direto do Pentaho, o Talend também oferece um ambiente de desenvolvimento gráfico para criar processos de integração de dados.
- **Vantagens:** Gera código Java ou SQL em segundo plano, o que pode resultar num melhor desempenho para certas tarefas. Possui uma vasta biblioteca de conectores.
- **Ideal para:** Projetos onde a performance e a escalabilidade são críticas, e para equipas com conhecimentos de Java que queiram estender as suas funcionalidades.

Ferramentas ETL com Interface Visual (GUI)

Apache NiFi:

- **Descrição:** Embora seja mais uma ferramenta de automação de fluxo de dados em tempo real, o NiFi é excelente para a parte de Extração e Carga (e transformações leves). A sua interface é baseada na web e focada no roteamento e processamento de *dataflows*.
- **Vantagens:** Ótimo para lidar com dados em *streaming* e fluxos contínuos. Oferece controlo total sobre a proveniência dos dados (*data lineage*).
- **Ideal para:** Casos de uso de IoT, processamento de *logs* e integração de sistemas em tempo real.

ETL com Programação (Código)

- Desenvolver processos de ETL através de código oferece máxima flexibilidade e controlo, sendo uma abordagem muito comum em equipas de engenharia de dados.

ETL com Programação (Código)

Python como Linguagem de ETL:

- **Porquê Python?** A sua sintaxe simples, ecossistema rico de bibliotecas e versatilidade tornaram-no a linguagem de eleição para a engenharia de dados.

ETL com Programação (Código)

Bibliotecas Essenciais:

- **Pandas:** A biblioteca fundamental para manipulação e análise de dados em Python. É perfeita para as fases de Extração (ler ficheiros CSV, JSON, Excel, etc.), Transformação (limpeza, junção, agregação, etc.) e Carga (escrever para ficheiros ou bases de dados).

ETL com Programação (Código)

- **SQLAlchemy:** Uma biblioteca que facilita a comunicação com bases de dados SQL. Permite carregar um *DataFrame* do Pandas diretamente para uma tabela de base de dados com poucas linhas de código.
- **Requests:** Essencial para extrair dados de APIs web.
- **PySpark:** Para cenários de *Big Data*, o *PySpark* permite usar o poder do *Apache Spark* para processar enormes volumes de dados de forma distribuída, usando a sintaxe familiar do Python.

Exemplo

- **Cenário:** Extrair dados de vendas (em CSV) e dados de clientes (em JSON), combiná-los para enriquecer a informação de vendas, e carregar o resultado final numa base de dados para análise.

- **Nota1: Há, propositadamente, um valor em falta (ID_Produto na venda 1007) para simular a necessidade de limpeza de dados.**

- **Nota2: O cliente C05 da venda 1008 não existe no ficheiro, simulando um problema de integridade referencial.**

Exemplo

Passos:

1. **Extrair** os dados de *vendas.csv* e *clientes.json*.

2. **Transformar** os dados:
   - Limpar vendas com dados em falta.
   - Juntar os dois conjuntos de dados.

3. **Carregar** o resultado num novo ficheiro CSV.

Exemplo


```
import pandas as pd #Importa a biblioteca Pandas
```

Exemplo


```
# --- 1. FASE DE EXTRAÇÃO (EXTRACT) ---

print("Iniciando o processo de ETL...")

# Extrair dados do ficheiro CSV de vendas para um DataFrame do Pandas
try:
    df_vendas = pd.read_csv('vendas.csv')              #Lê o ficheiro CSV e carrega-o para uma
    print("Ficheiro 'vendas.csv' lido com sucesso.")   #estrutura de dados chamada DataFrame, que é
except FileNotFoundError:                              #essencialmente uma tabela em memória.
    print("Erro: Ficheiro 'vendas.csv' não encontrado.")
    exit()

# Extrair dados do ficheiro JSON de clientes para um DataFrame do Pandas
try:
    df_clientes = pd.read_json('clientes.json')            #Faz o mesmo para o ficheiro JSON.
    print("Ficheiro 'clientes.json' lido com sucesso.")
except FileNotFoundError:
    print("Erro: Ficheiro 'clientes.json' não encontrado.")
    exit()
```

Exemplo


```python
# --- 2. FASE DE TRANSFORMAÇÃO (TRANSFORM) ---

print("\nIniciando a fase de transformação...")

# -- Limpeza de Dados --
# Verificar a quantidade de valores nulos antes da limpeza
print(f"Valores nulos em 'df_vendas' antes da limpeza:\n{df_vendas.isnull().sum()}\n")

# Remover linhas onde o 'ID_Produto' é nulo (equivalente ao "Filter Rows" do Pentaho)
df_vendas_limpo = df_vendas.dropna(subset=['ID_Produto'])  #Esta é a operação de limpeza. Remove todas as linhas do
print(f"Registos de vendas antes da limpeza: {len(df_vendas)}") #DataFrame df_vendas onde a coluna ID_Produto tem um valor nulo.
print(f"Registos de vendas após a limpeza: {len(df_vendas_limpo)}")

# -- Padronização de Nomes de Colunas --
# Renomear a coluna 'id_cliente' no DataFrame de clientes para corresponder à de vendas
# Isto facilita a operação de junção (merge)
df_clientes.rename(columns={'id_cliente': 'ID_Cliente'}, inplace=True) #Altera o nome da coluna id_cliente para ID_Cliente para que
print("\nColuna 'id_cliente' renomeada para 'ID_Cliente' no DataFrame de clientes.")   #ambas as tabelas tenham um nome de coluna comum para a junção.

# -- Junção de Dados (Join) --
# Juntar os DataFrames de vendas e clientes usando a coluna 'ID_Cliente'
# O 'how='inner'' garante que apenas as vendas com clientes correspondentes são mantidas
# (Equivalente ao "Merge Join" com INNER JOIN do Pentaho)
df_consolidado = pd.merge(   #Esta é a função principal de junção
    left=df_vendas_limpo,    #left e right...os dois DataFrames a serem unidos.
    right=df_clentes,
    on='ID_Cliente',         #especifica a coluna chave para a junção
    how='inner'              #define o tipo de junção. Um "inner join" inclui apenas as linhas onde a chave (ID_Cliente) existe em ambos os DataFrames.
)                            #Isto exclui automaticamente a venda do cliente C05, que não está no ficheiro de clientes

print("\nDados de vendas e clientes juntados com sucesso.")
print(f"Total de registos consolidados: {len(df_consolidado)}")
```

Exemplo


```python
# --- 3. FASE DE CARGA (LOAD) ---

print("\nIniciando a fase de carga...")

# Carregar o DataFrame consolidado para um novo ficheiro CSV
# O argumento index=False evita que o índice do DataFrame seja escrito no ficheiro
try:
    output_filename = 'vendas_consolidadas_python.csv'
    df_consolidado.to_csv(output_filename, index=False) #Guarda o conteúdo do DataFrame final num novo ficheiro CSV. index=False é importante para não
    print(f"Dados consolidados carregados com sucesso no ficheiro '{output_filename}'.")  #exportar uma coluna extra com os índices do DataFrame.
except Exception as e:
    print(f"Erro ao guardar o ficheiro de saída: {e}")

print("\nProcesso de ETL concluído!")

# Opcional: Mostrar as primeiras 5 linhas do resultado final
print("\nAmostra do resultado final:")
print(df_consolidado.head())
```

Python

- Python é uma linguagem de programação popular.

- Foi criado por Guido van Rossum e lançado em 1991.

- É usado para:
  - desenvolvimento web (lado do servidor),
  - desenvolvimento de software,
  - matemática,
  - script do sistema.

Python

- Linguagem de alto nível
- Sintaxe simples, fácil de aprender
- Implementação standard distribuída como código livre
- Suporta programação procedimental e orientada a objectos
- Muitas bibliotecas disponíveis
- Muito utilizada

Página web oficial: *http://www.python.org*

Python

- Os comentários podem ser usados para explicar o código Python.
- Os comentários podem ser usados para tornar o código mais legível.
- Os comentários podem ser usados para impedir a execução ao testar o código.
- Os comentários começam com #

Python

- Apenas tivemos que pedir para produzir/escrever/imprimir texto...

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

Python


```
Início
   Ler A
   Ler B
   C = A + B
   Escrever C
Fim
```

main.py ×

```python
#saudação
print('Olá')

#ler a
a = int(input("Escreva um número: "))

#ler b
b = int(input("Escreva outro número: "))

#Somar que é o meu "C"
c= a + b

#Escrever c
print("A soma é ", c)

```

Exercícios

Pedir dois números inteiros e indicar se são **iguais**. Caso os números sejam diferentes, não deve ser mostrada qualquer mensagem.

```
Introduz um número: 3
Introduz outro número: 3
Os números introduzidos são iguais.
```

```
print("----------------------")
print("Comparar dois números")
print("----------------------")

num1=int(input("\nEscreva um número: "))
num2=int(input("Escreva outro número: "))

print("")

if num1>num2:
    print("O primeiro número é superior ao segundo")
elif num1<num2:
    print("O segundo número é superior ao primeiro")
else:
    print("Os números são iguais")
```

Python


```
i = 1
while (i <= 9):
    print(i)
    i += 1
```

Python


```
x = range(9)

for i in x:
 print(i+1)
```

Ou...

```
x = range(1,10)

for i in x:
 print(i)
```

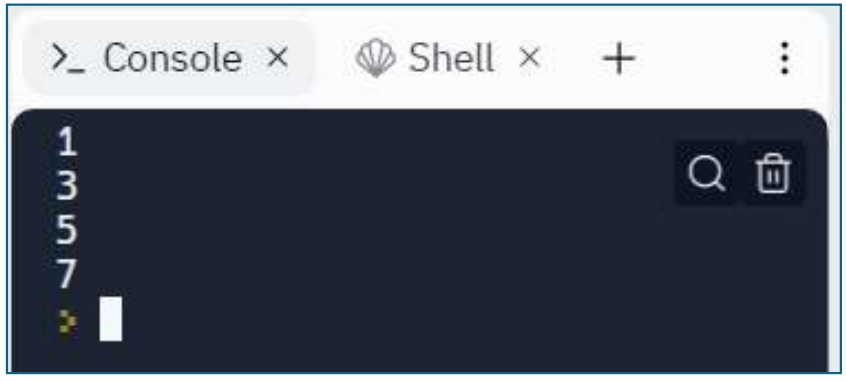
Python

A função **range()** retorna uma sequência de números, começando em 0 e incrementa 1 (por padrão), parando antes de um número especificado.

`range(start, stop, step)`

```
x = range(1,9,2)

for i in x:
 print(i)
```



Python


```
import random

for i in range(0, 5):
    print( random.randint(1, 50) )
```

```
30
19
50
4
39
```

Python


```
import random

x = 0
c = 0

while x != 50:
    x = random.randint(0, 100)
    c = c + 1

print(c)
```

```
23
```

Python


```
import random

x=0
c=0

while x!=50:
    x=random.randint(0,100)
    print("O valor de x é ", x)
    c=c+1

print("A quantidade de nºs gerados até sair o 50 foi ", c)
```

```
O valor de x é  66
O valor de x é  52
O valor de x é  15
O valor de x é  11
O valor de x é  47
O valor de x é  33
O valor de x é  70
O valor de x é  30
O valor de x é  1
O valor de x é  14
O valor de x é  17
O valor de x é  14
O valor de x é  59
O valor de x é  63
O valor de x é  50
A quantidade de nºs gerados até sair o 50 foi  15
```

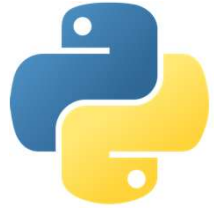
Exercícios

Criar um **dado** virtual, isto é, um programa que gere números entre 1 e 6.

```
import random

num=random.randint(1,6)

print("O nº saído após lançamento do dado foi:", num)
```



Exercícios

Modificar o programa anterior (dado virtual), de forma a **permitir lançamentos sucessivos**, isto é, com um funcionamento semelhante ao da simulação seguinte.

```
Lançar dado (S/N)? s
3
Lançar dado (S/N)? s
2
Lançar dado (S/N)? s
5
Lançar dado (S/N)? s
3
Lançar dado (S/N)? n
Terminado.
```

```
import random
resposta=input("Lançar dado (S/N)?")

while resposta=="S":

    num=random.randint(1,6)

    print("Dado lançado!")

    print("O nº saído após lançamento do dado foi:", num)

    resposta=input("Lançar dado (S/N)?")

print("Programa terminado!")
```