PYTHON ACADEMY

TRABALHANDO COM DATAS NO PYTHON

Neste ebook, abordaremos o módulo datetime do Python, explorando desde operações básicas até manipulações avançadas de datas e horários.

PYTHONACADEMY.COM.BR

> ✅ ***Atualizado para Python 3.13*** *(Fevereiro 2025)*
> *datetime, timedelta, timezone, strftime, strptime, casos práticos.*

Salve salve Pythonista 👋

Trabalhar com datas e horários é essencial em muitas aplicações Python.

Entender como manipular **datas** facilita tarefas como agendamentos, logs e análises temporais.

Neste artigo, abordaremos o módulo `datetime` do Python, explorando desde operações básicas até manipulações avançadas de datas e horários.

Com este conhecimento, você poderá gerenciar datas de forma eficiente em seus projetos Python.

Introdução ao Módulo `datetime`

O módulo `datetime` é a ferramenta padrão do Python para trabalhar com datas e horários.

Ele fornece classes para manipular datas, horários e intervalos de tempo.

Além disso, permite realizar operações como formatação, conversão e cálculo entre diferentes unidades de tempo.

Para começar, é necessário importar o módulo `datetime`:

```
import datetime
```

Trabalhando com Datas e Horas

Com o módulo `datetime`, podemos criar objetos que representam datas e horários específicos.

Vamos criar um objeto que representa a data atual:

```
data_atual = datetime.date.today()
print(data_atual)
```

Explicação do código:

1. **Importação**: Importamos o módulo `datetime`.
2. **Objeto de Data Atual**: Usamos `datetime.date.today()` para obter a data de hoje.
3. **Exibição**: Imprimimos a data atual no formato `YYYY-MM-DD`.

A saída será:

```
datetime.date(2025, 2, 25)
```

Criando um Objeto de Horário

Além de datas, podemos criar objetos que representam horários:

```
hora_atual = datetime.datetime.now().time()
print(hora_atual)
```

Explicação do código:

1. **Método `now()`**: `datetime.datetime.now()` retorna a data e hora atuais.
2. **Extração de Hora**: Usamos `.time()` para obter apenas o componente de horário.
3. **Exibição**: Imprimimos a hora atual no formato `HH:MM:SS.microsegundos`.

A saída será:

```
datetime.time(9, 27, 9, 484035)
```

Formatando Datas

Formatar datas é útil para exibir informações de maneira legível ou específica.

Usando `strftime`

A função `strftime` permite formatar objetos de data e hora:

```
data_atual = datetime.date.today()
data_formatada = data_atual.strftime("%d/%m/%Y")
print(data_formatada)
```

Explicação do código:

1. **Formato Personalizado**: `%d` representa o dia, `%m` o mês e `%Y` o ano.
2. **Formatação**: Convertendo `data_atual` para o formato `DD/MM/AAAA`.

3. **Exibição**: Imprimimos a data formatada.

E a saída será:

```
25/02/2025
```

Exemplo de Formatação Completa

```
hora_atual = datetime.datetime.now().time()
hora_formatada = hora_atual.strftime("%H:%M:%S")
print("Hora:", hora_formatada)
```

Explicação do código:

1. **Formato de Hora**: `%H` para horas, `%M` para minutos e `%S` para segundos.
2. **Formatação**: Convertendo `hora_atual` para o formato `HH:MM:SS`.
3. **Exibição**: Imprimimos a hora formatada.

A saída será:

```
Hora: 09:29:09
```

Convertendo Datas para Strings

Converter objetos de data para strings é útil para salvar ou exibir informações.

Usando `str()`

A função `str()` converte objetos de data para strings em um formato padrão:

```
data_atual = datetime.date.today()
data_str = str(data_atual)
print(data_str)
```

Explicação do código:

1. **Conversão**: Usamos `str(data_atual)` para converter o objeto de data em uma string.
2. **Exibição**: Imprimimos a string resultante no formato `YYYY-MM-DD`.

Sua saída será:

```
2025-02-25
```

Convertendo Strings para Datas

Às vezes, precisamos converter strings que representam datas em objetos de data.

Usando `strptime`

A função `strptime` permite analisar strings em objetos de data:

```
data_string = "25/12/2023"
data_convertida = datetime.datetime.strptime(data_string, "%d/%m/
        %Y").date()
print(data_convertida)
```

Explicação do código:

1. **String de Data**: Definimos `data_string` no formato `DD/MM/AAAA`.

2. **Análise**: Usamos `strptime` para converter a string em um objeto `datetime`.
3. **Extração de Data**: Com `.date()`, extraímos apenas a data.
4. **Exibição**: Imprimimos a data convertida no formato `YYYY-MM-DD`.

A saída será:

```
2023-12-25
```

Calculando Diferença entre Datas

Calcular a diferença entre datas é útil para determinar durações ou prazos.

Usando `timedelta`

A classe `timedelta` representa uma duração ou intervalo de tempo:

```
data_futura = datetime.date(2050, 1, 1)
diferenca = data_futura - data_atual
print(diferenca.days, "dias restantes")
```

Explicação do código:

1. **Data Futura**: Criamos uma data específica para o futuro.
2. **Cálculo da Diferença**: Subtraímos `data_atual` de `data_futura` para obter a diferença.
3. **Exibição**: Imprimimos o número de dias restantes.

A saída será:

```
9076 dias restantes
```

Trabalhando com Timezones

Gerenciar fusos horários é crucial para aplicações globais.

Usando `pytz`

Embora o módulo `datetime` tenha suporte básico, bibliotecas como `pytz` oferecem funcionalidades avançadas.

Para isso, instale o `pytz` utilizando o `pip` com `pip install pytz`. Em seguida:

```
import pytz

fuso_horario = pytz.timezone("America/Sao_Paulo")
data_hora_local = datetime.datetime.now(fuso_horario)
print(data_hora_local)
```

Explicação do código:

1. **Importação do `pytz`**: Importamos a biblioteca `pytz`.
2. **Definição do Fuso Horário**: Especificamos o fuso horário desejado.
3. **Obtendo Data e Hora Local**: Usamos `now` com o fuso horário definido.
4. **Exibição**: Imprimimos a data e hora local com o fuso aplicado.

A saída será:

```
2025-02-25 15:37:28.861810-03:00
```

A parte final da data `-03:00` é o fuso de São Paulo, GMT-3!

> ***Aproveitando o tema:*** *Estou desenvolvendo o* ***DevBook****, uma plataforma para criar ebooks técnicos com IA. Você escreve em Markdown, o sistema formata código com syntax highlighting e exporta PDFs profissionais. Ideal para documentar seu conhecimento. Dá uma olhada!*

Trabalhando com `timedelta`

A classe `timedelta` permite realizar operações com intervalos de tempo.

Adicionando Dias a uma Data

```
import datetime

data_atual = datetime.date.today()
intervalo = datetime.timedelta(days=10)
nova_data = data_atual + intervalo
print(nova_data)
```

Explicação do código:

1. **Criação do Intervalo**: Definimos um intervalo de 10 dias com `datetime.timedelta(days=10)`.
2. **Adição à Data Atual**: Somamos o intervalo à data atual.
3. **Exibição**: Imprimimos a nova data resultante.

A saída esperada é:

```
2025-03-07
```

Subtraindo Horas de um Horário

```
intervalo_horas = datetime.timedelta(hours=5)
nova_hora = datetime.datetime.now().time()
print("Hora atual:", nova_hora)

hora_ajustada = (datetime.datetime.combine(datetime.date.today(),
        nova_hora) - intervalo_horas).time()
print("Hora ajustada:", hora_ajustada)
```

Explicação do código:

1. **Criação do Intervalo de Horas**: Definimos um intervalo de 5 horas.
2. **Combinação de Data e Hora**: Combinamos a data de hoje com a hora atual.
3. **Subtração do Intervalo**: Subtraímos o intervalo de horas da data e hora combinadas.
4. **Extração da Hora Ajustada**: Obtemos apenas o componente de hora.
5. **Exibição**: Imprimimos a hora antes e após a subtração.

Veja a saída esperada:

```
Hora atual: 15:40:58.201792
Hora ajustada: 10:40:58.201792
```

Trabalhando com Datas em Diferentes Formatos

Datas podem estar em diversos formatos, dependendo da aplicação.

Parsing de Múltiplos Formatos

Podemos criar funções para lidar com múltiplos formatos de data:

```
def parse_data(data_str):
    for fmt in ("%d/%m/%Y", "%Y-%m-%d", "%m-%d-%Y"):
        try:
            return datetime.datetime.strptime(data_str, fmt).date()
        except ValueError:
            continue
    raise ValueError("Formato de data inválido")

data_exemplo = parse_data("12-31-2023")
print(data_exemplo)
```

Explicação do código:

1. **Função de Parsing**: Tentamos diferentes formatos para analisar a string de data.
2. **Laço de Tentativas**: Iteramos sobre uma lista de formatos possíveis.
3. **Retorno da Data**: Retornamos a data convertida se o formato corresponder.
4. **Exceção**: Lançamos um erro se nenhum formato for válido.
5. **Exibição**: Imprimimos a data convertida no formato `YYYY-MM-DD`.

Trabalhando com Datas em Diferentes Idiomas

Manipular datas em diferentes idiomas melhora a experiência do usuário global.

Usando `locale`

O módulo `locale` permite configurar a localização para formatação de datas:

```
import locale

# Seta a categoria do locale "tempo" (Locale Category: Time) para
# - Português do Brasil (pt_BR)
# - Codificação UTF-8
locale.setlocale(locale.LC_TIME, 'pt_BR.UTF-8')

data_atual = datetime.date.today()
data_formatada = data_atual.strftime("%A, %d de %B de %Y")
print(data_formatada)
```

Explicação do código:

1. **Importação do `locale`**: Importamos o módulo `locale`.
2. **Configuração da Localização**: Definimos a localidade para português brasileiro.
3. **Formatação com `locale`**: Usamos `strftime` para formatar a data com nomes de dias e meses em português.
4. **Exibição**: Imprimimos a data formatada, por exemplo, “segunda-feira, 25 de dezembro de 2023”.

A saída será:

```
terça, 25 de fevereiro de 2025
```

Casos Práticos Reais

1. Sistema de Agendamentos

```python
from datetime import datetime, timedelta
from zoneinfo import ZoneInfo

def agendar_reuniao(data_hora_str, duracao_minutos, timezone="America/
        Sao_Paulo"):
    """Agenda reunião e calcula horário de fim"""
    tz = ZoneInfo(timezone)
    inicio = datetime.fromisoformat(data_hora_str).replace(tzinfo=tz)
    fim = inicio + timedelta(minutes=duracao_minutos)

    return {
        "inicio": inicio.isoformat(),
        "fim": fim.isoformat(),
        "duracao": f"{duracao_minutos} minutos",
        "dia_semana": inicio.strftime("%A"),
        "horario": inicio.strftime("%H:%M")
    }

# Uso
reuniao = agendar_reuniao("2025-02-28 14:00:00", 90)
print(reuniao)
```

2. Cálculo de Idade e Validações

```python
from datetime import date
from dateutil.relativedelta import relativedelta

def calcular_idade_exata(data_nascimento):
    """Calcula idade exata em anos, meses e dias"""
    hoje = date.today()

    if isinstance(data_nascimento, str):
        data_nascimento = datetime.strptime(data_nascimento, "%Y-%m-
        %d").date()

    diff = relativedelta(hoje, data_nascimento)

    return {
        "anos": diff.years,
        "meses": diff.months,
        "dias": diff.days,
        "total_dias": (hoje - data_nascimento).days,
        "maior_idade": diff.years >= 18
    }

# Uso
idade = calcular_idade_exata("1990-05-15")
print(f"{idade['anos']} anos, {idade['meses']} meses, {idade['dias']}
        dias")
```

3. Dias Úteis (excluindo fins de semana)

```python
from datetime import date, timedelta

def calcular_dias_uteis(data_inicio, data_fim, feriados=None):
    """Calcula número de dias úteis entre duas datas"""
    if feriados is None:
        feriados = []

    dias_uteis = 0
    data_atual = data_inicio

    while data_atual <= data_fim:
        # 0=Segunda, 6=Domingo
        if data_atual.weekday() < 5 and data_atual not in feriados:
            dias_uteis += 1
        data_atual += timedelta(days=1)

    return dias_uteis

# Uso
inicio = date(2025, 2, 1)
fim = date(2025, 2, 28)
feriados = [date(2025, 2, 25)]  # Carnaval
uteis = calcular_dias_uteis(inicio, fim, feriados)
print(f"Dias úteis: {uteis}")
```

4. Lembretes e Notificações

```python
from datetime import datetime, timedelta

def calcular_lembretes(data_evento, lembretes=[7, 3, 1]):
    """Calcula datas para enviar lembretes antes do evento"""
    if isinstance(data_evento, str):
        data_evento = datetime.fromisoformat(data_evento)

    alertas = []
    for dias in lembretes:
        data_lembrete = data_evento - timedelta(days=dias)
        alertas.append({
            "dias_antes": dias,
            "data": data_lembrete.strftime("%d/%m/%Y"),
            "horario": data_lembrete.strftime("%H:%M"),
            "mensagem": f"Lembrete: evento em {dias} dia(s)"
        })

    return alertas

# Uso
event_date = "2025-03-15 10:00:00"
alertas = calcular_lembretes(event_date)
for alerta in alertas:
    print(f"{alerta['data']} - {alerta['mensagem']}")
```

Conclusão

Neste artigo, exploramos como **trabalhar com datas** no Python utilizando o módulo `datetime`.

Aprendemos a criar e formatar datas e horários, converter entre strings e objetos de data, calcular diferenças e manipular fusos horários.

Além disso, abordamos operações com `timedelta`, manipulações de datas futuras e passadas e formatação de datas em diferentes idiomas.

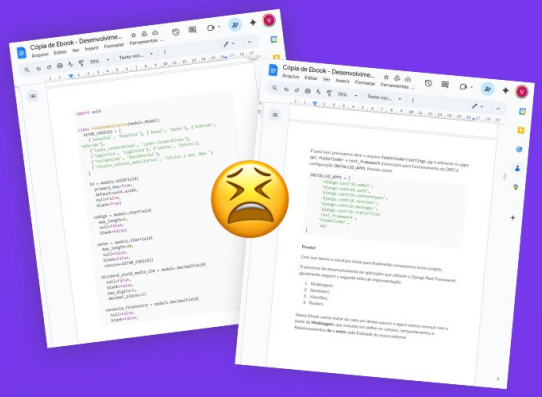
Com esses conhecimentos, você está preparado para gerenciar datas de forma eficaz em suas aplicações Python!

Não se esqueça de conferir!

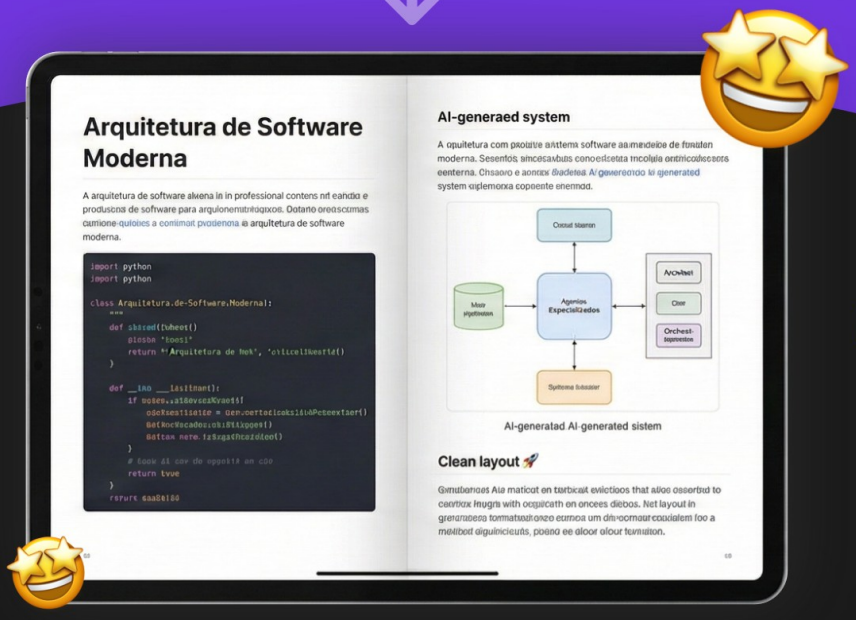
DevBook

Crie Ebooks técnicos em minutos com IA

Conheça a 1ª IA Especializada na criação de Ebooks **com código**!



Chega de formatar código no Google Docs



Syntax Highlight

Infográficos feitos por IA

Adicione Banners Promocionais

Deixe que nossa IA faça o trabalho pesado

Edite em Markdown em Tempo Real

TESTE AGORA

PRIMEIRO CAPÍTULO 100% GRÁTIS