合肥大學
HEFEI UNIVERSITY

Programming with Python

30. Funktionen als Parameter, `Callables` und `lambdas`

Thomas Weise (汤卫思)
tweise@hfuu.edu.cn

Institute of Applied Optimization (IAO)
School of Artificial Intelligence and Big Data
Hefei University
Hefei, Anhui, China

应用优化研究所
人工智能与大数据学院
合肥大学
中国安徽省合肥市

Version: 2026-01-29

Programming with Python

Dies ist ein Kurs über das Programmieren mit der Programmiersprache Python an der Universität Hefei (合肥大学).

Die Webseite mit dem Lehrmaterial dieses Kurses ist https://thomasweise.github.io/programmingWithPython (siehe auch den QR-Kode unten rechts). Dort können Sie das Kursbuch (in Englisch) und diese Slides finden. Das Repository mit den Beispielprogrammen in Python finden Sie unter https://github.com/thomasWeise/programmingWithPythonCode.

Outline

Einleitung

Einleitung

- Wir haben gelernt, wie man Funktionen definiert und aufruft.

Einleitung

- Wir haben gelernt, wie man Funktionen definiert und aufruft.
- Wir haben gelernt, wie Funktionen Ergebnisse zurückliefern können.

Einleitung

- Wir haben gelernt, wie man Funktionen definiert und aufruft.
- Wir haben gelernt, wie Funktionen Ergebnisse zurückliefern können.
- Wir haben gelernt, wie man Funktionen testet.

Einleitung

- Wir haben gelernt, wie man Funktionen definiert und aufruft.
- Wir haben gelernt, wie Funktionen Ergebnisse zurückliefern können.
- Wir haben gelernt, wie man Funktionen testet.
- Und wir haben gerade eben gelernt, wie man Funktionsaufrufe konstruieren kann, in dem man die Parameterwerte in einer Kollektion speichert und diese dann beim Aufruf in die Parameterliste „entpackt“.

Einleitung

- Wir haben gelernt, wie man Funktionen definiert und aufruft.
- Wir haben gelernt, wie Funktionen Ergebnisse zurückliefern können.
- Wir haben gelernt, wie man Funktionen testet.
- Und wir haben gerade eben gelernt, wie man Funktionsaufrufe konstruieren kann, in dem man die Parameterwerte in einer Kollektion speichert und diese dann beim Aufruf in die Parameterliste „entpackt“.
- Es gibt noch eine interessante Sache, die wir in Python mit Funktionen machen können.

Einleitung

- Wir haben gelernt, wie man Funktionen definiert und aufruft.
- Wir haben gelernt, wie Funktionen Ergebnisse zurückliefern können.
- Wir haben gelernt, wie man Funktionen testet.
- Und wir haben gerade eben gelernt, wie man Funktionsaufrufe konstruieren kann, in dem man die Parameterwerte in einer Kollektion speichert und diese dann beim Aufruf in die Parameterliste „entpackt“.
- Es gibt noch eine interessante Sache, die wir in Python mit Funktionen machen können.
- In Python sind alle Dinge Objekte[20,43].

Einleitung

- Wir haben gelernt, wie man Funktionen definiert und aufruft.
- Wir haben gelernt, wie Funktionen Ergebnisse zurückliefern können.
- Wir haben gelernt, wie man Funktionen testet.
- Und wir haben gerade eben gelernt, wie man Funktionsaufrufe konstruieren kann, in dem man die Parameterwerte in einer Kollektion speichert und diese dann beim Aufruf in die Parameterliste „entpackt“.
- Es gibt noch eine interessante Sache, die wir in Python mit Funktionen machen können.
- In Python sind alle Dinge Objekte[20,43].
- Referenzen auf Objekte können in Variablen gespeichert werden oder als Parameter an Funktionen übergeben werden.

Einleitung

- Wir haben gelernt, wie man Funktionen definiert und aufruft.
- Wir haben gelernt, wie Funktionen Ergebnisse zurückliefern können.
- Wir haben gelernt, wie man Funktionen testet.
- Und wir haben gerade eben gelernt, wie man Funktionsaufrufe konstruieren kann, in dem man die Parameterwerte in einer Kollektion speichert und diese dann beim Aufruf in die Parameterliste „entpackt“.
- Es gibt noch eine interessante Sache, die wir in Python mit Funktionen machen können.
- In Python sind alle Dinge Objekte[20,43].
- Referenzen auf Objekte können in Variablen gespeichert werden oder als Parameter an Funktionen übergeben werden.
- Funktionen selber sind auch Objekte[20].

Einleitung

- Wir haben gelernt, wie man Funktionen definiert und aufruft.
- Wir haben gelernt, wie Funktionen Ergebnisse zurückliefern können.
- Wir haben gelernt, wie man Funktionen testet.
- Und wir haben gerade eben gelernt, wie man Funktionsaufrufe konstruieren kann, in dem man die Parameterwerte in einer Kollektion speichert und diese dann beim Aufruf in die Parameterliste „entpackt“.
- Es gibt noch eine interessante Sache, die wir in Python mit Funktionen machen können.
- In Python sind alle Dinge Objekte[20,43].
- Referenzen auf Objekte können in Variablen gespeichert werden oder als Parameter an Funktionen übergeben werden.
- Funktionen selber sind auch Objekte[20].
- Das bedeutet, dass wir Funktionen in Variablen speichern oder sie als Argumente an andere Funktionen übergeben können.

Einleitung

- Wir haben gelernt, wie Funktionen Ergebnisse zurückliefern können.
- Wir haben gelernt, wie man Funktionen testet.
- Und wir haben gerade eben gelernt, wie man Funktionsaufrufe konstruieren kann, in dem man die Parameterwerte in einer Kollektion speichert und diese dann beim Aufruf in die Parameterliste „entpackt“.
- Es gibt noch eine interessante Sache, die wir in Python mit Funktionen machen können.
- In Python sind alle Dinge Objekte[20,43].
- Referenzen auf Objekte können in Variablen gespeichert werden oder als Parameter an Funktionen übergeben werden.
- Funktionen selber sind auch Objekte[20].
- Das bedeutet, dass wir Funktionen in Variablen speichern oder sie als Argumente an andere Funktionen übergeben können.
- Das klingt erstmal komisch.

Einleitung

- Wir haben gelernt, wie man Funktionen testet.
- Und wir haben gerade eben gelernt, wie man Funktionsaufrufe konstruieren kann, in dem man die Parameterwerte in einer Kollektion speichert und diese dann beim Aufruf in die Parameterliste „entpackt“.
- Es gibt noch eine interessante Sache, die wir in Python mit Funktionen machen können.
- In Python sind alle Dinge Objekte[20,43].
- Referenzen auf Objekte können in Variablen gespeichert werden oder als Parameter an Funktionen übergeben werden.
- Funktionen selber sind auch Objekte[20].
- Das bedeutet, dass wir Funktionen in Variablen speichern oder sie als Argumente an andere Funktionen übergeben können.
- Das klingt erstmal komisch.
- Warum würde man eine Funktion als Parameter an eine andere Funktion übergeben wollen?

Einleitung

- Und wir haben gerade eben gelernt, wie man Funktionsaufrufe konstruieren kann, in dem man die Parameterwerte in einer Kollektion speichert und diese dann beim Aufruf in die Parameterliste „entpackt“.
- Es gibt noch eine interessante Sache, die wir in Python mit Funktionen machen können.
- In Python sind alle Dinge Objekte[20,43].
- Referenzen auf Objekte können in Variablen gespeichert werden oder als Parameter an Funktionen übergeben werden.
- Funktionen selber sind auch Objekte[20].
- Das bedeutet, dass wir Funktionen in Variablen speichern oder sie als Argumente an andere Funktionen übergeben können.
- Das klingt erstmal komisch.
- Warum würde man eine Funktion als Parameter an eine andere Funktion übergeben wollen?
- Auf dem zweiten Blick fallen uns durchaus ein paar Anwendungen ein.

Einleitung

- Es gibt noch eine interessante Sache, die wir in Python mit Funktionen machen können.
- In Python sind alle Dinge Objekte[20,43].
- Referenzen auf Objekte können in Variablen gespeichert werden oder als Parameter an Funktionen übergeben werden.
- Funktionen selber sind auch Objekte[20].
- Das bedeutet, dass wir Funktionen in Variablen speichern oder sie als Argumente an andere Funktionen übergeben können.
- Das klingt erstmal komisch.
- Warum würde man eine Funktion als Parameter an eine andere Funktion übergeben wollen?
- Auf dem zweiten Blick fallen uns durchaus ein paar Anwendungen ein.
- Schauen wir uns eine aus der Mathematik an.

Beispielszenario: Bestimmtes Integral

Bestimmtes Integral

- In der Schule haben Sie Differential- und Integralrechnung gelernt.

Bestimmtes Integral

- In der Schule haben Sie Differential- und Integralrechnung gelernt.
- Ein *bestimmtes Integral* berechnet die Fläche unter einer Funktion über einem bestimmten Abschnitt der x-Achse.

Bestimmtes Integral

- In der Schule haben Sie Differential- und Integralrechnung gelernt.
- Ein *bestimmtes Integral* berechnet die Fläche unter einer Funktion über einem bestimmten Abschnitt der x-Achse.
- Dazu werden infinitesimal-kleine Streifen über diesem Abschnitt verwendet.

Bestimmtes Integral

- In der Schule haben Sie Differential- und Integralrechnung gelernt.
- Ein *bestimmtes Integral* berechnet die Fläche unter einer Funktion über einem bestimmten Abschnitt der x-Achse.
- Dazu werden infinitesimal-kleine Streifen über diesem Abschnitt verwendet.
- Infinitesimal bedeutet „*unermesslich klein*“ oder „*beliebig nahe an aber größer als 0*“[27,49].

Bestimmtes Integral

- In der Schule haben Sie Differential- und Integralrechnung gelernt.
- Ein *bestimmtes Integral* berechnet die Fläche unter einer Funktion über einem bestimmten Abschnitt der x-Achse.
- Dazu werden infinitesimal-kleine Streifen über diesem Abschnitt verwendet.
- Infinitesimal bedeutet *„unermesslich klein“* oder *„beliebig nahe an aber größer als 0“*[27,49].
- Der Hauptsatz der Differential- und Integralrechnung sagt aus[3,101]: *Wenn eine Funktion $f(x)$ über dem Intervall $[a, b]$ kontinuierlich ist und $F(x)$ die Stammfunktion von $f(x)$ ist (also $F' = f$), dann ist das bestimmte Integral $\int_a^b f(x)\,dx = F(b) - F(a)$.*

Bestimmtes Integral

- In der Schule haben Sie Differential- und Integralrechnung gelernt.
- Ein *bestimmtes Integral* berechnet die Fläche unter einer Funktion über einem bestimmten Abschnitt der x-Achse.
- Dazu werden infinitesimal-kleine Streifen über diesem Abschnitt verwendet.
- Infinitesimal bedeutet *„unermesslich klein“* oder *„beliebig nahe an aber größer als 0“*[27,49].
- Der Hauptsatz der Differential- und Integralrechnung sagt aus[3,101]: *Wenn eine Funktion $f(x)$ über dem Intervall $[a, b]$ kontinuierlich ist und $F(x)$ die Stammfunktion von $f(x)$ ist (also $F' = f$), dann ist das bestimmte Integral $\int_a^b f(x)\,dx = F(b) - F(a)$.*
- Mit anderen Worten, wenn wir die Fläche unter $f(x)$ über dem Intervall $[a, b]$ bestimmen wollen, dann würden wir erst die Stammfunktion $F(x)$ herleiten und dann einfach $F(b) - F(a)$ berechnen.

Bestimmtes Integral

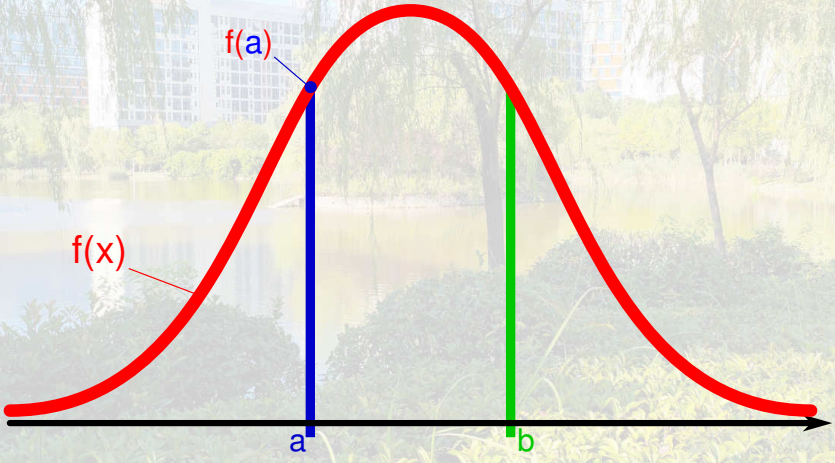
- In der Schule haben Sie Differential- und Integralrechnung gelernt.
- Ein *bestimmtes Integral* berechnet die Fläche unter einer Funktion über einem bestimmten Abschnitt der x-Achse.
- Dazu werden infinitesimal-kleine Streifen über diesem Abschnitt verwendet.
- Infinitesimal bedeutet *„unermesslich klein“* oder *„beliebig nahe an aber größer als 0“*[27,49].
- Der Hauptsatz der Differential- und Integralrechnung sagt aus[3,101]: *Wenn eine Funktion $f(x)$ über dem Intervall $[a, b]$ kontinuierlich ist und $F(x)$ die Stammfunktion von $f(x)$ ist (also $F' = f$), dann ist das bestimmte Integral $\int_a^b f(x)\,dx = F(b) - F(a)$.*
- Mit anderen Worten, wenn wir die Fläche unter $f(x)$ über dem Intervall $[a, b]$ bestimmen wollen, dann würden wir erst die Stammfunktion $F(x)$ herleiten und dann einfach $F(b) - F(a)$ berechnen.
- Die Stammfunktion können wir auf einen Stück Papier mit symbolischer Mathematik herleiten.

Bestimmtes Integral

- In der Schule haben Sie Differential- und Integralrechnung gelernt.
- Ein *bestimmtes Integral* berechnet die Fläche unter einer Funktion über einem bestimmten Abschnitt der x-Achse.
- Dazu werden infinitesimal-kleine Streifen über diesem Abschnitt verwendet.
- Infinitesimal bedeutet *„unermesslich klein“* oder *„beliebig nahe an aber größer als 0“*[27,49].
- Der Hauptsatz der Differential- und Integralrechnung sagt aus[3,101]: *Wenn eine Funktion $f(x)$ über dem Intervall $[a, b]$ kontinuierlich ist und $F(x)$ die Stammfunktion von $f(x)$ ist (also $F' = f$), dann ist das bestimmte Integral $\int_a^b f(x)\,dx = F(b) - F(a)$.*
- Mit anderen Worten, wenn wir die Fläche unter $f(x)$ über dem Intervall $[a, b]$ bestimmen wollen, dann würden wir erst die Stammfunktion $F(x)$ herleiten und dann einfach $F(b) - F(a)$ berechnen.
- Die Stammfunktion können wir auf einen Stück Papier mit symbolischer Mathematik herleiten.
- So etwas können wir an diese Stelle nicht wirklich mit Python programmieren.

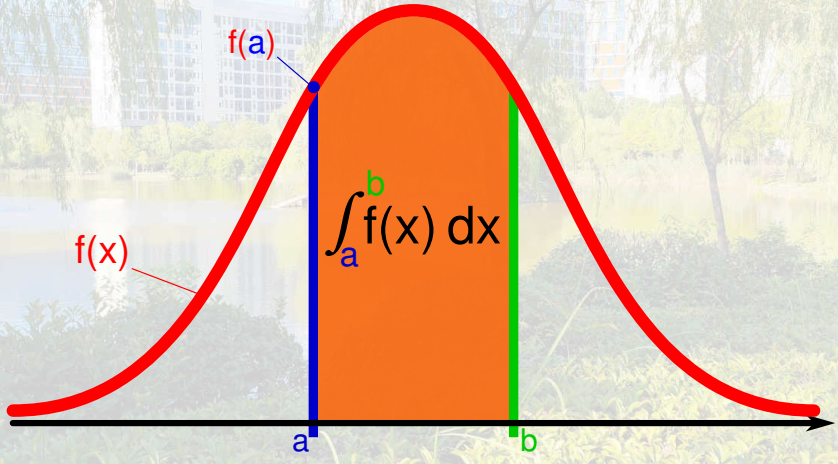
Annäherung des Bestimmten Integrals

- Wir können zur Definition des bestimmten Integrals zurückkehren, nämlich der Fläche unter Funktion über dem Intervall $[a, b]$.



Annäherung des Bestimmten Integrals

- Wir können zur Definition des bestimmten Integrals zurückkehren, nämlich der Fläche unter Funktion über dem Intervall $[a, b]$.



Annäherung des Bestimmten Integrals

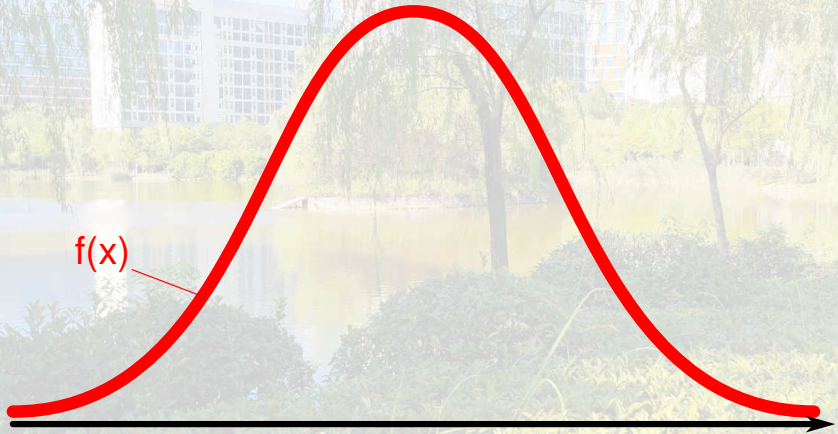
- Wir können zur Definition des bestimmten Integrals zurückkehren, nämlich der Fläche unter Funktion über dem Intervall $[a, b]$.
- Und diese Fläche kann über infinitesimal kleine Streifen über dem Intervall berechnet werden.

Annäherung des Bestimmten Integrals

- Wir können zur Definition des bestimmten Integrals zurückkehren, nämlich der Fläche unter Funktion über dem Intervall $[a, b]$.
- Und diese Fläche kann über infinitesimal kleine Streifen über dem Intervall berechnet werden.
- Nun können wir nicht wirklich „*unermesslich kleine*“ Streifen machen . . . aber wir könnten zumindest *sehr kleine* Streifen machen, um das richtige Ergebnis zumindest anzunähern.

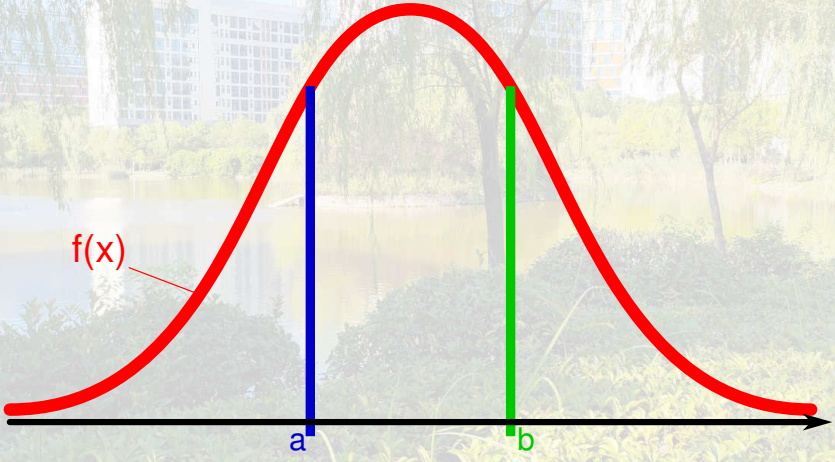
Annäherung des Bestimmten Integrals

- Sagen wir, dass wir die Funktion $f(x)$ als Parameter für unser Programm haben.



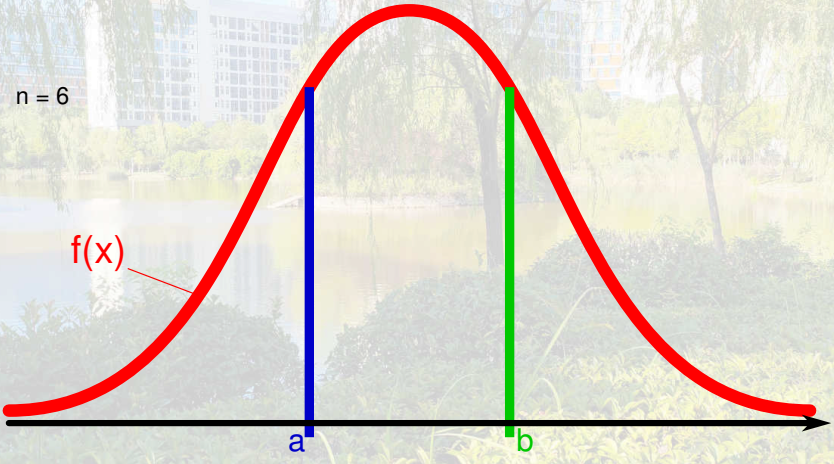
Annäherung des Bestimmten Integrals

- Sagen wir, dass wir die Funktion $f(x)$ als Parameter für unser Programm haben.
- Die Intervallgrenzen a und b wären ebenfalls Parameter.



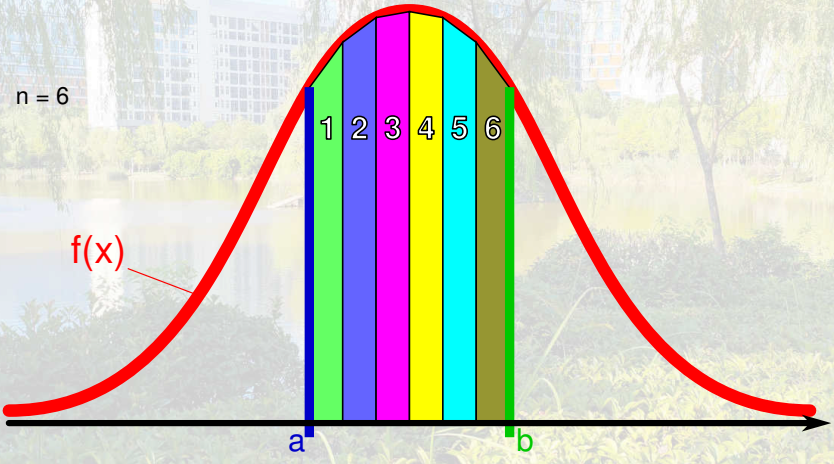
Annäherung des Bestimmten Integrals

- Die Intervallgrenzen a und b wären ebenfalls Parameter.
- Dann könnten wir das Intervall $[a, b]$ in n Steifen einteilen.



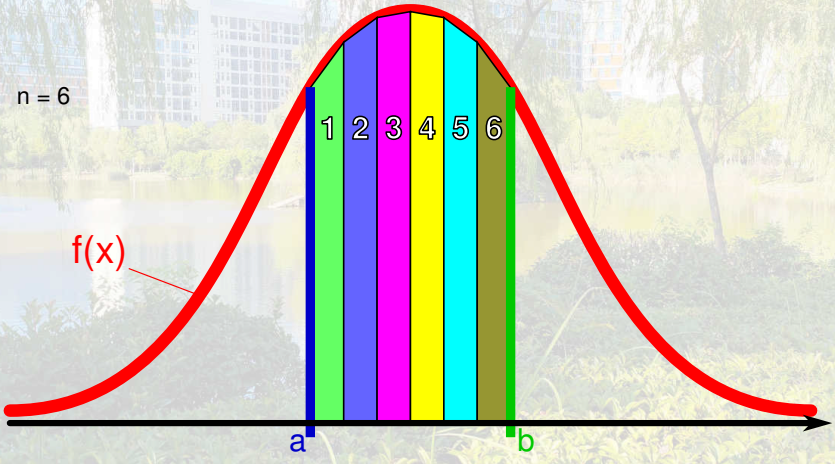
Annäherung des Bestimmten Integrals

- Dann könnten wir das Intervall $[a, b]$ in n Steifen einteilen.
- Für jeden Streifen würden wir die Fläche unter $f(x)$ annähern.



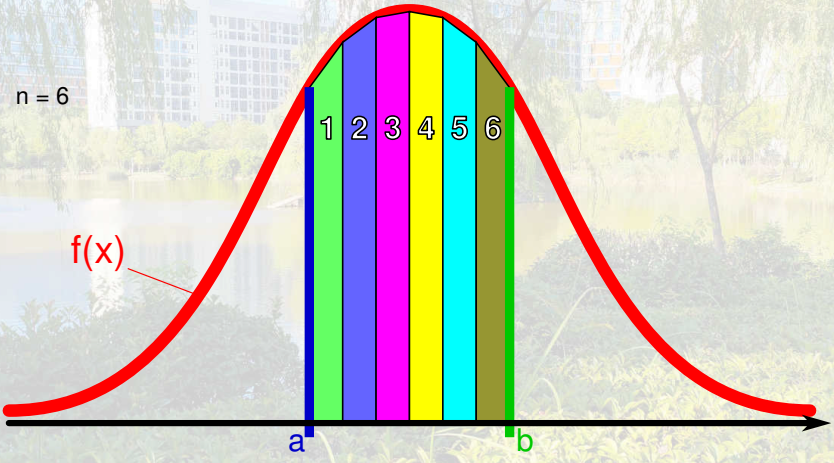
Annäherung des Bestimmten Integrals

- Für jeden Streifen würden wir die Fläche unter $f(x)$ annähern.
- Dann summieren wir die n Flächen auf und haben eine Annäherung der Gesamtfläche.



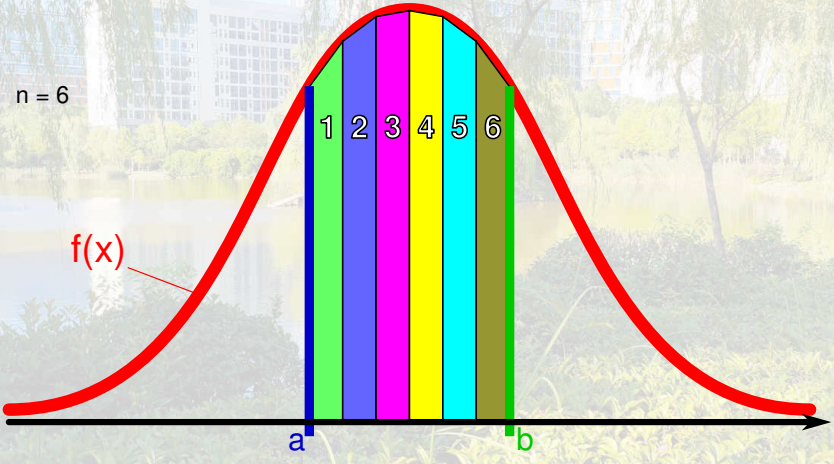
Annäherung des Bestimmten Integrals

- Dann summieren wir die n Flächen auf und haben eine Annäherung der Gesamtfläche.
- n könnte auch ein Parameter unseres Programms sein.



Annäherung des Bestimmten Integrals

- Je größer n, desto schmaler werden die Steifen, desto genauer wird unsere Annäherung (und desto länger dauert es, sie zu berechnen).



Trapez-Methode

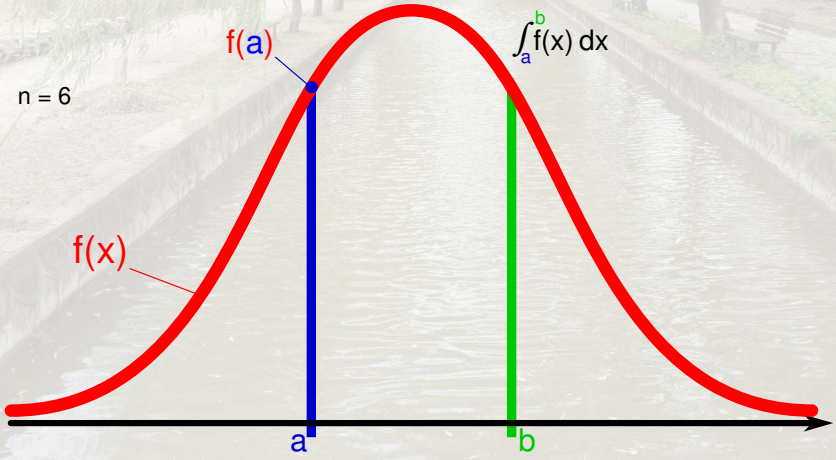
- Aber wie geht das? Und wie können wir die Fläche eines Streifens der Funktion überhaupt annähern?

Trapez-Methode

- Aber wie geht das? Und wie können wir die Fläche eines Streifens der Funktion überhaupt annähern?
- Die Trapez-Methode ist eine sehr einfache Implementierung der Idee[6,28,47].

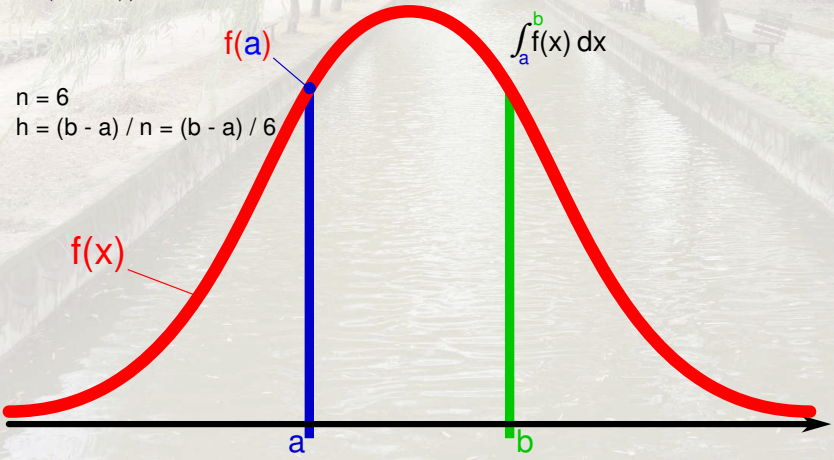
Trapez-Methode

- Die Trapez-Methode ist eine sehr einfache Implementierung der Idee[6,28,47].
- Sie behandelt die Streifen als Trapeze.



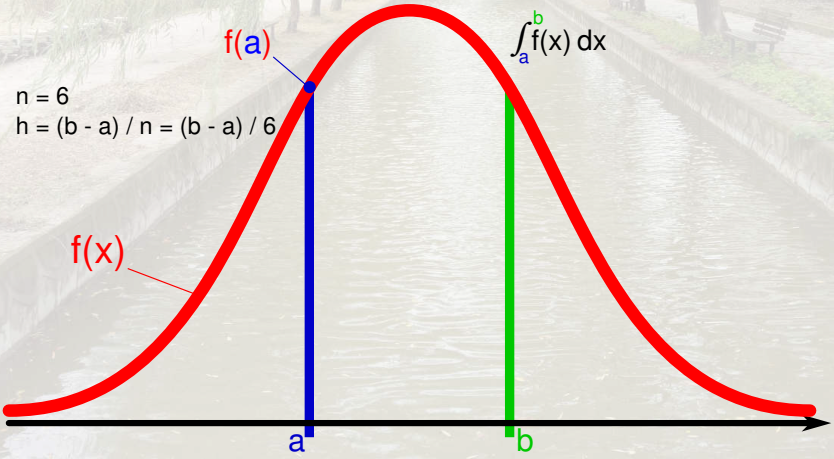
Trapez-Methode

- Sie behandelt die Streifen als Trapeze.
- Die Grundlinie der Trapeze ist dabei jeweils ein Stück der x-Achse mit Länge $h = (b - a)/n$.



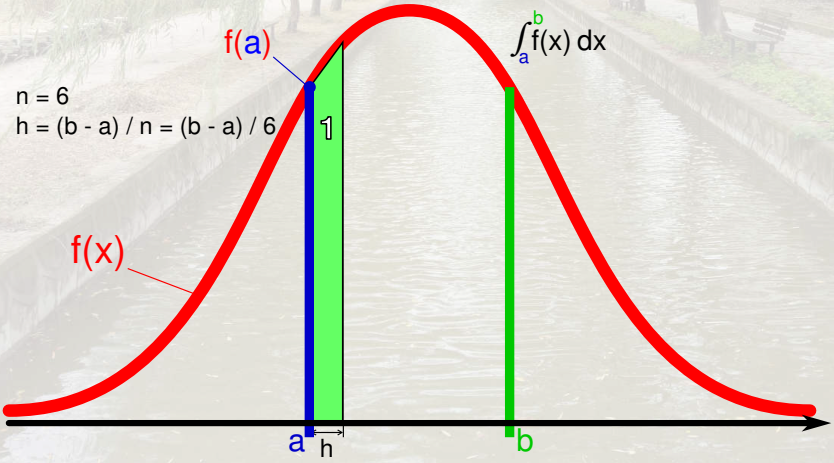
Trapez-Methode

- Offensichtlich ist $n * h = b - a$ und darum ergeben n Trapeze von gleicher Grundlinienlänge ergeben das Interval $[a, b]$ auf der x-Achse unter $f(x)$.



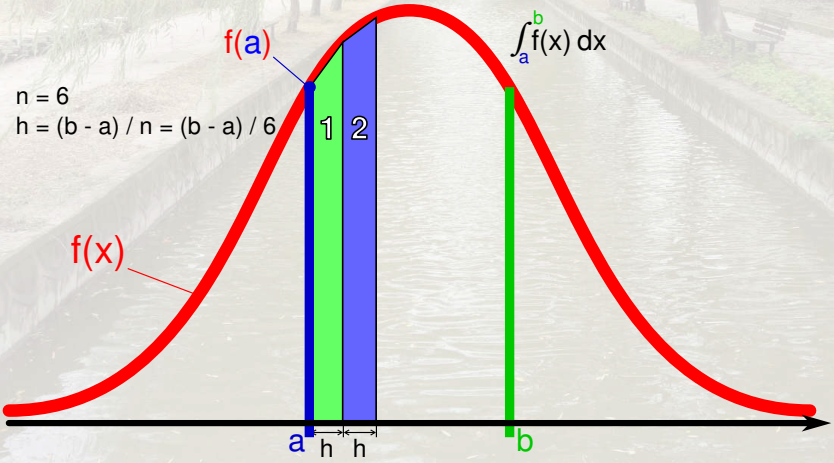
Trapez-Methode

- Dafür startet das erste Trapez bei $x = a$ und endet an $x = a + h$.



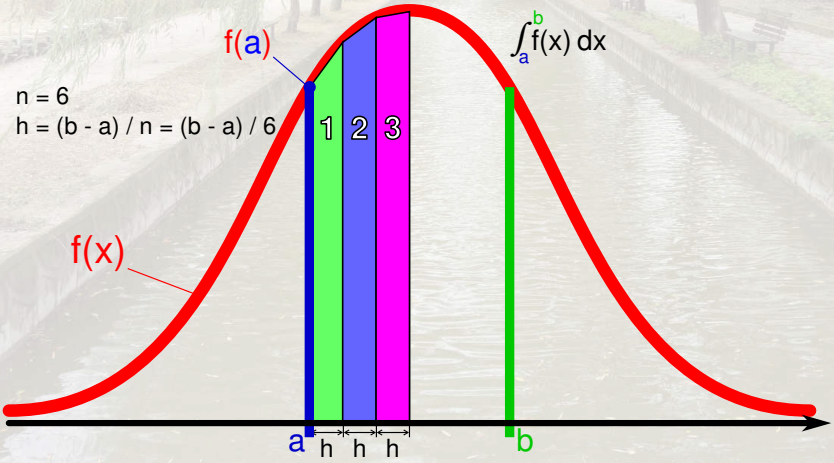
Trapez-Methode

- Dafür startet das erste Trapez bei $x = a$ und endet an $x = a + h$.
- Das zweite Trapez startet bei $x = a + h$ und endet bei $x = a + 2h$.



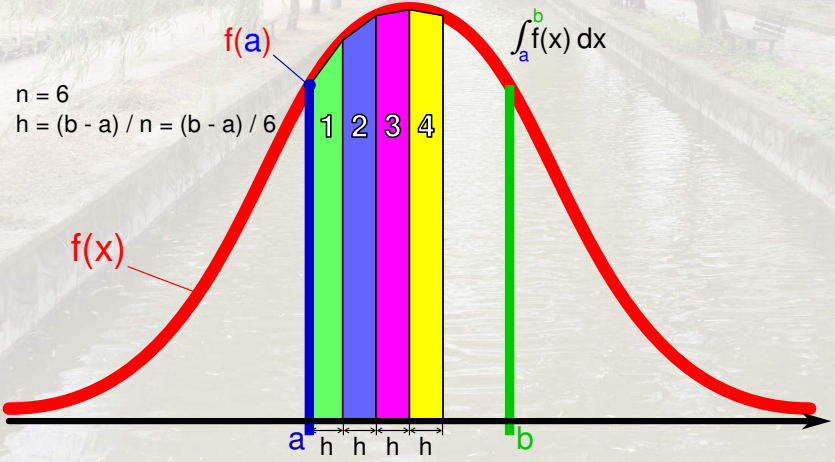
Trapez-Methode

- Das zweite Trapez startet bei $x = a + h$ und endet bei $x = a + 2h$.
- So geht es immer weiter.



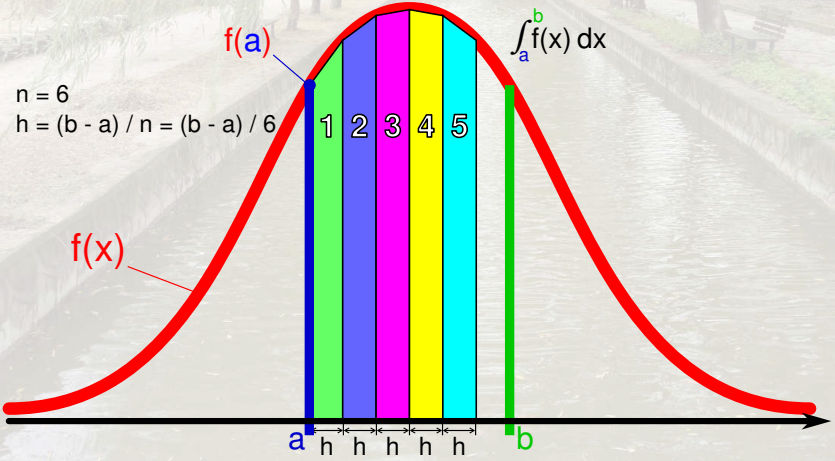
Trapez-Methode

- Das zweite Trapez startet bei $x = a + h$ und endet bei $x = a + 2h$.
- So geht es immer weiter.



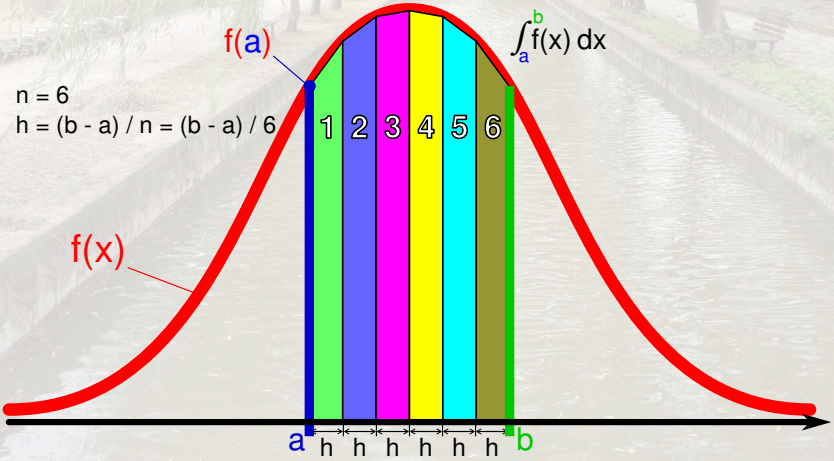
Trapez-Methode

- Das zweite Trapez startet bei $x = a + h$ und endet bei $x = a + 2h$.
- So geht es immer weiter.



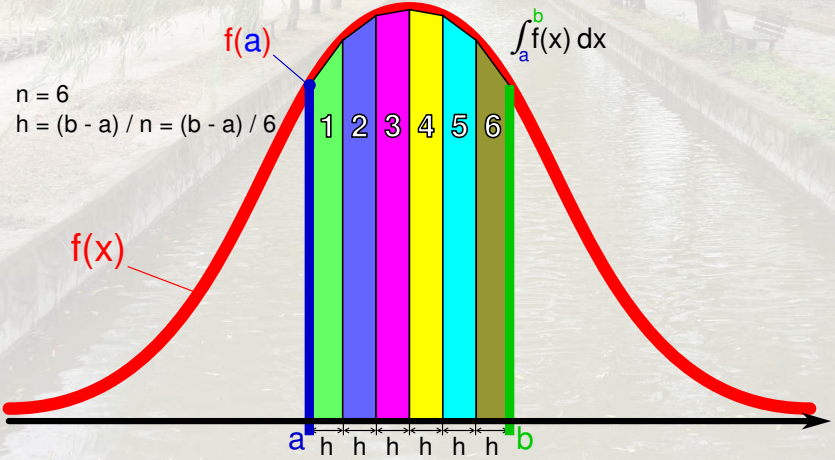
Trapez-Methode

- So geht es immer weiter.
- Das letzte Trapez startet bei $x = a + (n-1)h$ und endet bei $x = a + nh = b$.



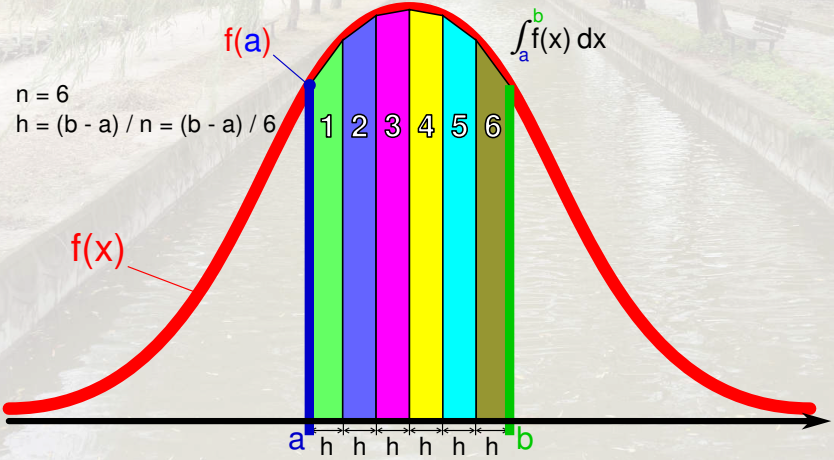
Trapez-Methode

- Das letzte Trapez startet bei $x = a + (n-1)h$ und endet bei $x = a + nh = b$.
- Jedes Trapez hat zwei parallele Seiten, die die Grundlinie jeweils im rechten Winkel treffen.



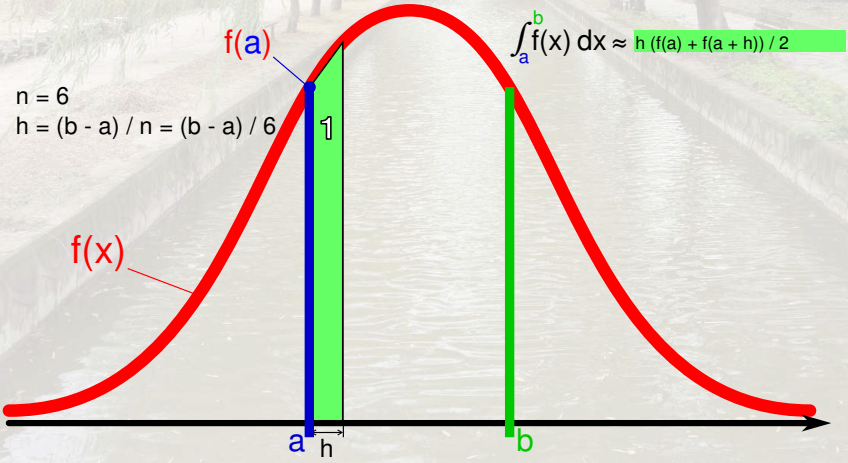
Trapez-Methode

- Jedes Trapez hat zwei parallele Seiten, die die Grundlinie jeweils im rechten Winkel treffen.
- Die Länge der Seiten entspricht $f(x)$ an den entsprechenden x-Koordinaten.



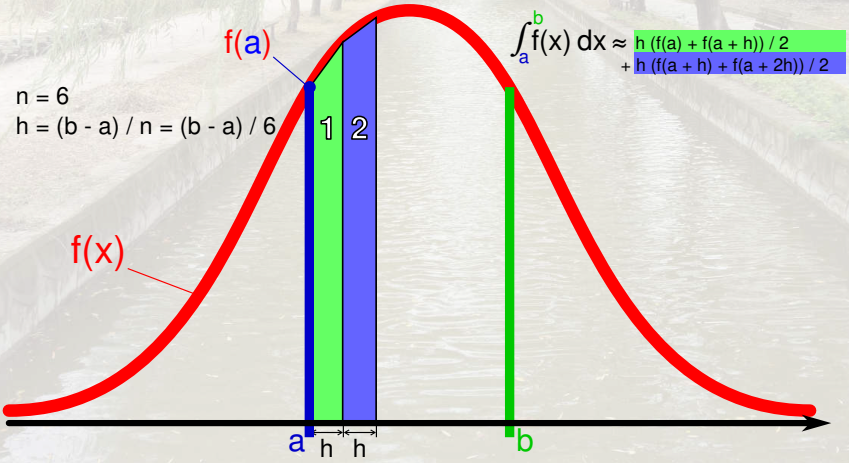
Trapez-Methode

- Die Länge der Seiten entspricht $f(x)$ an den entsprechenden x-Koordinaten.
- Die Fläche des i-ten Trapez ist deshalb $h[f(a + (i-1)h) + f(a + ih)]/2$.



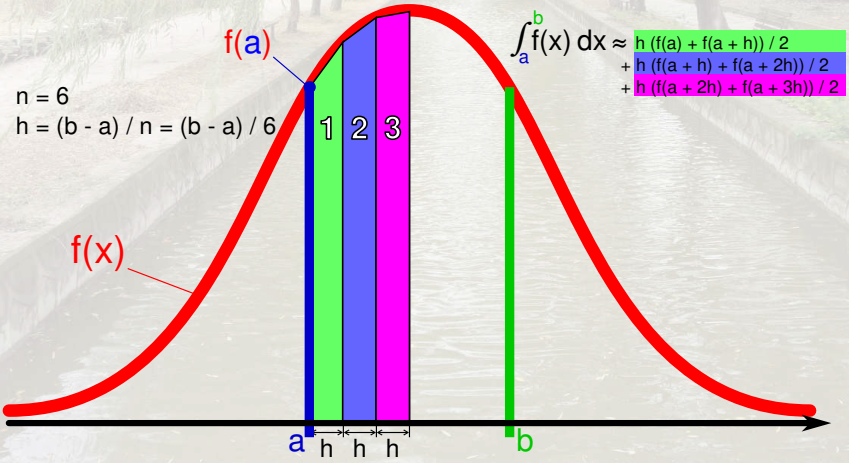
Trapez-Methode

- Die Fläche des i-ten Trapez ist deshalb $h[f(a+(i-1)h)+f(a+ih)]/2$.



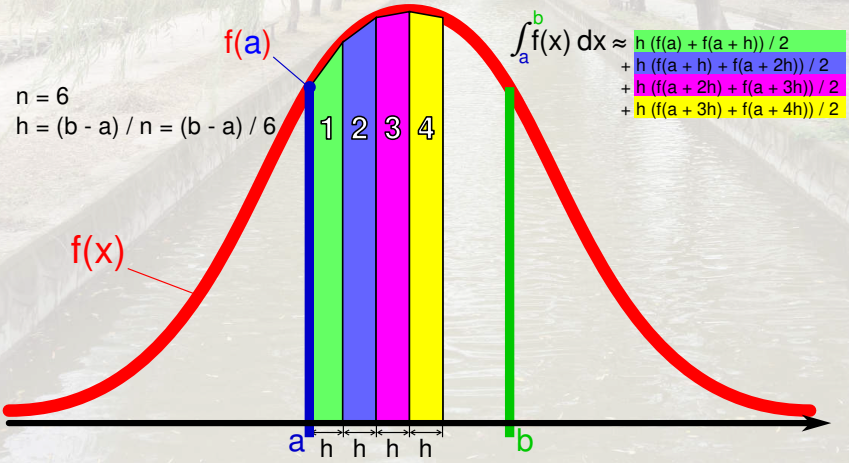
Trapez-Methode

- Die Fläche des i-ten Trapez ist deshalb $h[f(a+(i-1)h)+f(a+ih)]/2$.



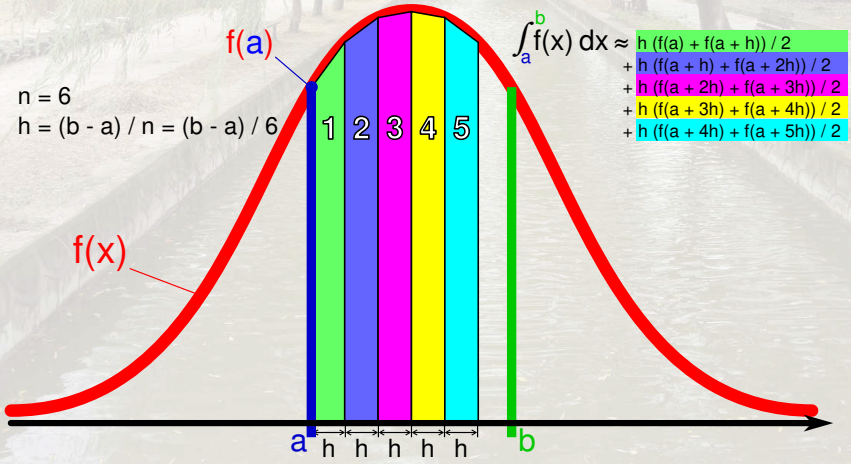
Trapez-Methode

- Die Fläche des i-ten Trapez ist deshalb $h[f(a+(i-1)h)+f(a+ih)]/2$.



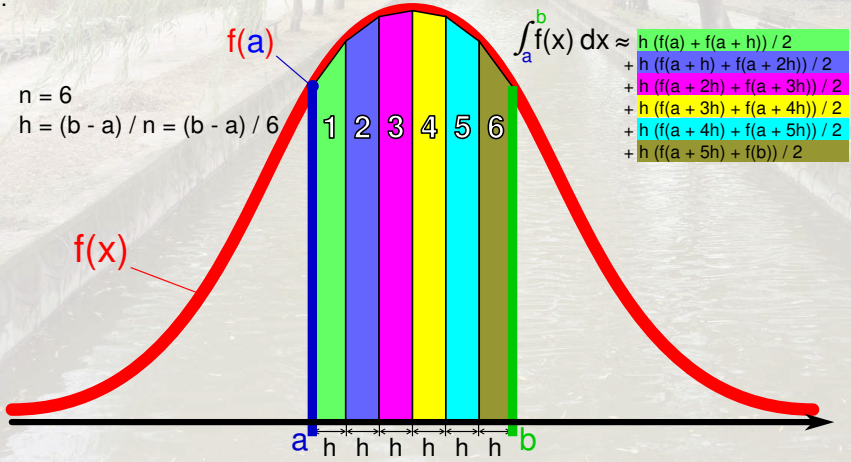
Trapez-Methode

- Die Fläche des i-ten Trapez ist deshalb $h[f(a+(i-1)h)+f(a+ih)]/2$.



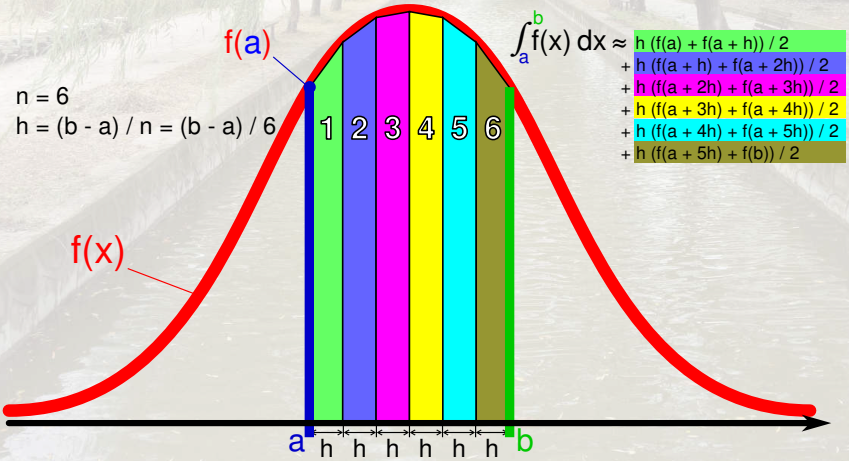
Trapez-Methode

- Die Fläche des i-ten Trapez ist deshalb $h[f(a+(i-1)h)+f(a+ih)]/2$.
- Wenn wir alle n Flächen aufaddieren haben wir eine Annäherung für das bestimmte Integral.



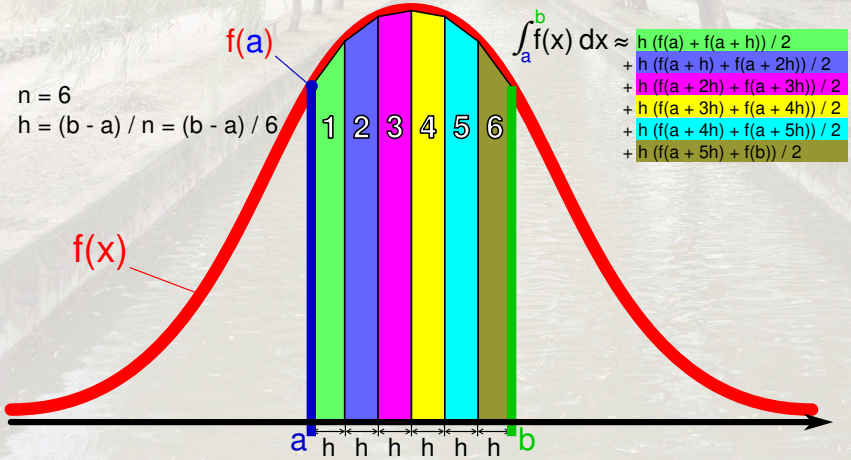
Trapez-Methode

- Dabei kommt jeder Wert $f(a+ih)$ zweimal vor, außer für $i=0$ und $i=n$.



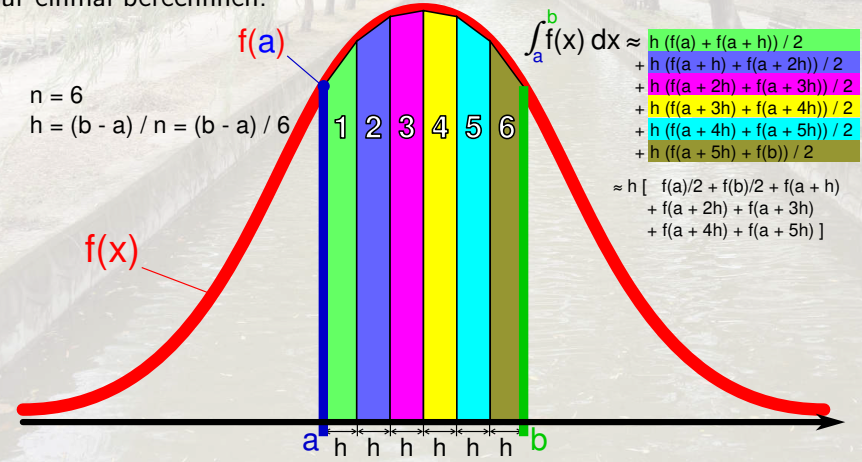
Trapez-Methode

- Dabei kommt jeder Wert $f(a+ih)$ zweimal vor, außer für $i=0$ und $i=n$.
- Die Werte werden bei der Flächenberechnung jeweils halbiert.



Trapez-Methode

- Die Werte werden bei der Flächenberechnung jeweils halbiert.
- Anstatt die doppelten Werte zweimal zu berechnen und dann zu halbieren, können wir sie auch nur einmal berechnnen.



$$\int_a^b f(x)\,dx \approx$$

h (f(a) + f(a + h)) / 2
+ h (f(a + h) + f(a + 2h)) / 2
+ h (f(a + 2h) + f(a + 3h)) / 2
+ h (f(a + 3h) + f(a + 4h)) / 2
+ h (f(a + 4h) + f(a + 5h)) / 2
+ h (f(a + 5h) + f(b)) / 2

≈ h [f(a)/2 + f(b)/2 + f(a + h)
+ f(a + 2h) + f(a + 3h)
+ f(a + 4h) + f(a + 5h)]

Implementierung

Implementierung

- Jetzt wollen wir diese Methode als Funktion `integrate` implementieren.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

Implementierung

- Jetzt wollen wir diese Methode als Funktion `integrate` implementieren.
- Offensichtlich braucht `integrate` die Parameter `a` und `b`, die die Grenzen des Intervalls, über das wir integrieren wollen, festlegen.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

Implementierung

- Jetzt wollen wir diese Methode als Funktion `integrate` implementieren.
- Offensichtlich braucht `integrate` die Parameter `a` und `b`, die die Grenzen des Intervalls, über das wir integrieren wollen, festlegen.
- Wir erlauben, dass diese entweder `int` oder `float`s seien können, was wir durch den Type Hint `float | int` ausdrücken.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

Implementierung

- Jetzt wollen wir diese Methode als Funktion `integrate` implementieren.
- Offensichtlich braucht `integrate` die Parameter `a` und `b`, die die Grenzen des Intervalls, über das wir integrieren wollen, festlegen.
- Wir erlauben, dass diese entweder `int` oder `float`s seien können, was wir durch den Type Hint `float | int` ausdrücken.
- Wir geben ihnen die Default Values `0.0` und `1.0`.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

Implementierung

- Jetzt wollen wir diese Methode als Funktion `integrate` implementieren.
- Offensichtlich braucht `integrate` die Parameter `a` und `b`, die die Grenzen des Intervalls, über das wir integrieren wollen, festlegen.
- Wir erlauben, dass diese entweder `int` oder `floats` seien können, was wir durch den Type Hint `float | int` ausdrücken.
- Wir geben ihnen die Default Values `0.0` und `1.0`.
- Dann ist da auch der Parameter `n`, ein `int`, der die Anzahl der Trapeze definiert.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

Implementierung

- Offensichtlich braucht `integrate` die Parameter `a` und `b`, die die Grenzen des Intervalls, über das wir integrieren wollen, festlegen.
- Wir erlauben, dass diese entweder `int` oder `floats` seien können, was wir durch den Type Hint `float | int` ausdrücken.
- Wir geben ihnen die Default Values `0.0` und `1.0`.
- Dann ist da auch der Parameter `n`, ein `int`, der die Anzahl der Trapeze definiert.
- Ein guter Default Value für `n` ist vielleicht `100`.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

Implementierung

- Wir erlauben, dass diese entweder `int` oder `floats` seien können, was wir durch den Type Hint `float | int` ausdrücken.
- Wir geben ihnen die Default Values `0.0` und `1.0`.
- Dann ist da auch der Parameter `n`, ein `int`, der die Anzahl der Trapeze definiert.
- Ein guter Default Value für `n` ist vielleicht `100`.
- Wir brauchen aber einen anderen Parameter, nämlich die Funktion `f`, die wir ja integrieren wollen.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

Implementierung

- Wir geben ihnen die Default Values `0.0` und `1.0`.
- Dann ist da auch der Parameter `n`, ein `int`, der die Anzahl der Trapeze definiert.
- Ein guter Default Value für `n` ist vielleicht `100`.
- Wir brauchen aber einen anderen Parameter, nämlich die Funktion `f`, die wir ja integrieren wollen.
- Wie können wir diesen Parameter spezifizieren?

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

Implementierung

- Dann ist da auch der Parameter `n`, ein `int`, der die Anzahl der Trapeze definiert.
- Ein guter Default Value für `n` ist vielleicht `100`.
- Wir brauchen aber einen anderen Parameter, nämlich die Funktion `f`, die wir ja integrieren wollen.
- Wie können wir diesen Parameter spezifizieren?
- Als `f: Callable[[float], float |int]`

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

Implementierung

- Ein guter Default Value für `n` ist vielleicht `100`.
- Wir brauchen aber einen anderen Parameter, nämlich die Funktion `f`, die wir ja integrieren wollen.
- Wie können wir diesen Parameter spezifizieren?
- Als `f: Callable[[float], float |int]`
- `Callable` ist ein Type Hint für alles was aufgerufen werden kann, wie z. B. Funktionen[2].

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

Implementierung

- Wir brauchen aber einen anderen Parameter, nämlich die Funktion `f`, die wir ja integrieren wollen.
- Wie können wir diesen Parameter spezifizieren?
- Als `f: Callable[[float], float |int]`
- `Callable` ist ein Type Hint für alles was aufgerufen werden kann, wie z. B. Funktionen[2].
- Wie die Type Hints für Listen und Tupel kann er mit eckigen Klammern parameterisiert werden[61].

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

```python
"""The syntax of type hints for callable objects like functions."""

Callable[[parameterType1, parameterType2, ...], resultType]
```

Implementierung

- Wie können wir diesen Parameter spezifizieren?
- Als `f: Callable[[float], float |int]`
- `Callable` ist ein Type Hint für alles was aufgerufen werden kann, wie z. B. Funktionen[2].
- Wie die Type Hints für Listen und Tupel kann er mit eckigen Klammern parameterisiert werden[61].
- In dem `Callable[...]` geben wir zuest die Liste der Typen der Parameter der Funktion an, wieder in eckigen Klammern.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

```python
"""The syntax of type hints for callable objects like functions."""

Callable[[parameterType1, parameterType2, ...], resultType]
```

Implementierung

- Als `f: Callable[[float], float |int]`
- `Callable` ist ein Type Hint für alles was aufgerufen werden kann, wie z. B. Funktionen[2].
- Wie die Type Hints für Listen und Tupel kann er mit eckigen Klammern parameterisiert werden[61].
- In dem `Callable[...]` geben wir zuest die Liste der Typen der Parameter der Funktion an, wieder in eckigen Klammern.
- Dann kommt ein Komma `,` und dann der Typ des Rückgabewerts.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

```python
"""The syntax of type hints for callable objects like functions."""

Callable[[parameterType1, parameterType2, ...], resultType]
```

Implementierung

- Wie die Type Hints für Listen und Tupel kann er mit eckigen Klammern parameterisiert werden[61].
- In dem `Callable[...]` geben wir zuest die Liste der Typen der Parameter der Funktion an, wieder in eckigen Klammern.
- Dann kommt ein Komma `,` und dann der Typ des Rückgabewerts.
- Das `f: Callable[[float], float |int]` bedeutet also das unsere Funktion `integrate` eine andere Funktion `f` als ersten Parameter erwartet.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

```python
"""The syntax of type hints for callable objects like functions."""

Callable[[parameterType1, parameterType2, ...], resultType]
```

Implementierung

- Wie die Type Hints für Listen und Tupel kann er mit eckigen Klammern parameterisiert werden[61].
- In dem `Callable[...]` geben wir zuest die Liste der Typen der Parameter der Funktion an, wieder in eckigen Klammern.
- Dann kommt ein Komma `,` und dann der Typ des Rückgabewerts.
- Das `f: Callable[[float], float |int]` bedeutet also das unsere Funktion `integrate` eine andere Funktion `f` als ersten Parameter erwartet.
- `f` muss genau einen Parameter erwarten, und zwar vom Typ `float`.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

```python
"""The syntax of type hints for callable objects like functions."""

Callable[[parameterType1, parameterType2, ...], resultType]
```

Implementierung

- Dann kommt ein Komma `,` und dann der Typ des Rückgabewerts.
- Das `f: Callable[[float], float |int]` bedeutet also das unsere Funktion `integrate` eine andere Funktion `f` als ersten Parameter erwartet.
- `f` muss genau einen Parameter erwarten, und zwar vom Typ `float`.
- Der Typ des Rückgabewerts von `f` ist `float |int`, also soll es entweder einen `float` oder einen `int` zurückliefern[72].

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

```python
"""The syntax of type hints for callable objects like functions."""

Callable[[parameterType1, parameterType2, ...], resultType]
```

Implementierung

- Das `f: Callable[[float], float |int]` bedeutet also das unsere Funktion `integrate` eine andere Funktion `f` als ersten Parameter erwartet.
- `f` muss genau einen Parameter erwarten, und zwar vom Typ `float`.
- Der Typ des Rückgabewerts von `f` ist `float |int`, also soll es entweder einen `float` oder einen `int` zurückliefern[72].
- Der Typ `Callable` wird aus dem Modul `typing` importiert, kann aber auch aus `collections.abc.Callable` importiert werden[2].

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

```python
"""The syntax of type hints for callable objects like functions."""

Callable[[parameterType1, parameterType2, ...], resultType]
```

Implementierung

- `f` muss genau einen Parameter erwarten, und zwar vom Typ `float`.
- Der Typ des Rückgabewerts von `f` ist `float | int`, also soll es entweder einen `float` oder einen `int` zurückliefern[72].
- Der Typ `Callable` wird aus dem Modul `typing` importiert, kann aber auch aus `collections.abc.Callable` importiert werden[2].
- Die Implementierung von `integrate` ist recht einfach.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

```python
"""The syntax of type hints for callable objects like functions."""

Callable[[parameterType1, parameterType2, ...], resultType]
```

Implementierung

- Der Typ des Rückgabewerts von `f` ist `float | int`, also soll es entweder einen `float` oder einen `int` zurückliefern[72].
- Der Typ `Callable` wird aus dem Modul `typing` importiert, kann aber auch aus `collections.abc.Callable` importiert werden[2].
- Die Implementierung von `integrate` ist recht einfach.
- In der Funktion addieren wir die Trapeze aus der vereinfachten Gleichung ohne Doppelungen zusammen.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

```python
"""The syntax of type hints for callable objects like functions."""

Callable[[parameterType1, parameterType2, ...], resultType]
```

Implementierung

- Der Typ `Callable` wird aus dem Modul `typing` importiert, kann aber auch aus `collections.abc.Callable` importiert werden[2].
- Die Implementierung von `integrate` ist recht einfach.
- In der Funktion addieren wir die Trapeze aus der vereinfachten Gleichung ohne Doppelungen zusammen.
- Nur die Werte von `f` an den Enden des Intervalls, nämlich bei `a` und `b`, müssen halbiert werden.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

```python
"""The syntax of type hints for callable objects like functions."""

Callable[[parameterType1, parameterType2, ...], resultType]
```

Implementierung

- Der Typ `Callable` wird aus dem Modul `typing` importiert, kann aber auch aus `collections.abc.Callable` importiert werden[2].
- Die Implementierung von `integrate` ist recht einfach.
- In der Funktion addieren wir die Trapeze aus der vereinfachten Gleichung ohne Doppelungen zusammen.
- Nur die Werte von `f` an den Enden des Intervalls, nämlich bei `a` und `b`, müssen halbiert werden.
- Wir initialisieren die Summe `result` als `0.5 * (f (a)+ f(b))`.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

```python
"""The syntax of type hints for callable objects like functions."""

Callable[[parameterType1, parameterType2, ...], resultType]
```

Implementierung

- Die Implementierung von `integrate` ist recht einfach.
- In der Funktion addieren wir die Trapeze aus der vereinfachten Gleichung ohne Doppelungen zusammen.
- Nur die Werte von `f` an den Enden des Intervalls, nämlich bei `a` und `b`, müssen halbiert werden.
- Wir initialisieren die Summe `result` als `0.5 * (f (a)+ f(b))`.
- Wir berechnen die Basislänge `h` der Trapeze als `(b - a)/ n`.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

```python
"""The syntax of type hints for callable objects like functions."""

Callable[[parameterType1, parameterType2, ...], resultType]
```

Implementierung

- In der Funktion addieren wir die Trapeze aus der vereinfachten Gleichung ohne Doppelungen zusammen.
- Nur die Werte von `f` an den Enden des Intervalls, nämlich bei `a` und `b`, müssen halbiert werden.
- Wir initialisieren die Summe `result` als `0.5 * (f (a)+ f(b))`.
- Wir berechnen die Basislänge `h` der Trapeze als `(b - a) / n`.
- Dann iterieren wir den Zähler `i` von `1` bis `n - 1` und addieren `f(a + h * i)` zu `result` in jedem Schritt.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

```python
"""The syntax of type hints for callable objects like functions."""

Callable[[parameterType1, parameterType2, ...], resultType]
```

Implementierung

- Nur die Werte von `f` an den Enden des Intervalls, nämlich bei `a` und `b`, müssen halbiert werden.
- Wir initialisieren die Summe `result` als `0.5 * (f (a)+ f(b))`.
- Wir berechnen die Basislänge `h` der Trapeze als `(b - a) / n`.
- Dann iterieren wir den Zähler `i` von `1` bis `n - 1` und addieren `f(a + h * i)` zu `result` in jedem Schritt.
- Am Ende müssen wir diesen Wert mit der Basislänge `h` der Trapeze multiplizieren, um die näherungsweise Fläche unter der Kurve zu bekommen.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

```python
"""The syntax of type hints for callable objects like functions."""

Callable[[parameterType1, parameterType2, ...], resultType]
```

Implementierung

- Wir initialisieren die Summe `result` als `0.5 * (f (a)+ f(b))`.
- Wir berechnen die Basislänge `h` der Trapeze als `(b - a) / n`.
- Dann iterieren wir den Zähler `i` von `1` bis `n - 1` und addieren `f(a + h * i)` zu `result` in jedem Schritt.
- Am Ende müssen wir diesen Wert mit der Basislänge `h` der Trapeze multiplizieren, um die näherungsweise Fläche unter der Kurve zu bekommen.
- Wir liefern also `result * h` zurück.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

```python
"""The syntax of type hints for callable objects like functions."""

Callable[[parameterType1, parameterType2, ...], resultType]
```

Implementierung

- Wir berechnen die Basislänge `h` der Trapeze als `(b - a) / n`.
- Dann iterieren wir den Zähler `i` von `1` bis `n - 1` und addieren `f(a + h * i)` zu `result` in jedem Schritt.
- Am Ende müssen wir diesen Wert mit der Basislänge `h` der Trapeze multiplizieren, um die näherungsweise Fläche unter der Kurve zu bekommen.
- Wir liefern also `result * h` zurück.
- Nun wollen nun endlich ausprobieren, wir unsere Trapez-basierte Integration funktioniert.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

```python
"""The syntax of type hints for callable objects like functions."""

Callable[[parameterType1, parameterType2, ...], resultType]
```

Implementierung

- Dann iterieren wir den Zähler `i` von `1` bis `n - 1` und addieren `f(a + h * i)` zu `result` in jedem Schritt.
- Am Ende müssen wir diesen Wert mit der Basislänge `h` der Trapeze multiplizieren, um die näherungsweise Fläche unter der Kurve zu bekommen.
- Wir liefern also `result * h` zurück.
- Nun wollen nun endlich ausprobieren, wir unsere Trapez-basierte Integration funktioniert.
- Berechnen wir also erstmal das bestimmte Integral $\int_0^1 1\,dx$.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

```python
"""The syntax of type hints for callable objects like functions."""

Callable[[parameterType1, parameterType2, ...], resultType]
```

Implementierung

- Am Ende müssen wir diesen Wert mit der Basislänge `h` der Trapeze multiplizieren, um die näherungsweise Fläche unter der Kurve zu bekommen.
- Wir liefern also `result * h` zurück.
- Nun wollen nun endlich ausprobieren, wir unsere Trapez-basierte Integration funktioniert.
- Berechnen wir also erstmal das bestimmte Integral $\int_0^1 1\,dx$.
- Dafür müssen wir die Funktion $f(x) = 1$ als Parameter `f` an `integrate` übergeben.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

```python
"""The syntax of type hints for callable objects like functions."""

Callable[[parameterType1, parameterType2, ...], resultType]
```

Implementierung

- Wir liefern also `result * h` zurück.
- Nun wollen nun endlich ausprobieren, wir unsere Trapez-basierte Integration funktioniert.
- Berechnen wir also erstmal das bestimmte Integral $\int_0^1 1\,dx$.
- Dafür müssen wir die Funktion $f(x) = 1$ als Parameter `f` an `integrate` übergeben.
- Wir könnten dazu folgendes schreiben:

```python
"""The syntax of a unary function always returning a constance value."""
def const_1(x: float) -> float:
    return 1.0  # always return 1.0, regardless of the value of x

# or

# It is common to use "_" as name for parameters that we will ignore.
def const_1(_: float) -> float:
    return 1.0  # always return 1.0, parameter "_" is ignored.
```

Implementierung

- Nun wollen nun endlich ausprobieren, wir unsere Trapez-basierte Integration funktioniert.
- Berechnen wir also erstmal das bestimmte Integral $\int_0^1 1\,dx$.
- Dafür müssen wir die Funktion $f(x) = 1$ als Parameter `f` an `integrate` übergeben.
- Wir könnten dazu folgendes schreiben:
- Hier drückt der Unterstrich `_` aus das wir den Parameter der Funktion ignorieren werden.

```python
"""The syntax of a unary function always returning a constance value."""
def const_1(x: float) -> float:
    return 1.0  # always return 1.0, regardless of the value of x

# or

# It is common to use "_" as name for parameters that we will ignore.
def const_1(_: float) -> float:
    return 1.0  # always return 1.0, parameter "_" is ignored.
```

Implementierung

- Berechnen wir also erstmal das bestimmte Integral $\int_0^1 1\,dx$.
- Dafür müssen wir die Funktion $f(x) = 1$ als Parameter `f` an `integrate` übergeben.
- Wir könnten dazu folgendes schreiben:
- Hier drückt der Unterstrich `_` aus das wir den Parameter der Funktion ignorieren werden.
- Wir könnten nun `integrate(const_1)` aufrufen.

```python
"""The syntax of a unary function always returning a constance value."""
def const_1(x: float) -> float:
    return 1.0  # always return 1.0, regardless of the value of x

# or

# It is common to use "_" as name for parameters that we will ignore.
def const_1(_: float) -> float:
    return 1.0  # always return 1.0, parameter "_" is ignored.
```

Implementierung

- Dafür müssen wir die Funktion $f(x) = 1$ als Parameter `f` an `integrate` übergeben.
- Wir könnten dazu folgendes schreiben:
- Hier drückt der Unterstrich `_` aus das wir den Parameter der Funktion ignorieren werden.
- Wir könnten nun `integrate(const_1)` aufrufen.
- Es gibt aber einen besseren, bequemeren Weg, Funktionen zu spezifizieren, die wir nur einmal verwenden wollen.

```python
"""The syntax of a unary function always returning a constance value."""
def const_1(x: float) -> float:
    return 1.0  # always return 1.0, regardless of the value of x

# or

# It is common to use "_" as name for parameters that we will ignore.
def const_1(_: float) -> float:
    return 1.0  # always return 1.0, parameter "_" is ignored.
```

Implementierung

- Wir könnten dazu folgendes schreiben:
- Hier drückt der Unterstrich `_` aus das wir den Parameter der Funktion ignorieren werden.
- Wir könnten nun `integrate(const_1)` aufrufen.
- Es gibt aber einen besseren, bequemeren Weg, Funktionen zu spezifizieren, die wir nur einmal verwenden wollen.
- Dieser Weg sind die sogenannten `lambdas`[51].

```python
"""The syntax of a unary function always returning a constance value."""
def const_1(x: float) -> float:
    return 1.0  # always return 1.0, regardless of the value of x

# or

# It is common to use "_" as name for parameters that we will ignore.
def const_1(_: float) -> float:
    return 1.0  # always return 1.0, parameter "_" is ignored.
```

```python
"""The syntax of lambdas, i.e., inline function definitions."""

# This defines a nameless inline function with two parameters that
# returns "return_value".
lambda param1, param2: return_value
```

Implementierung

- Hier drückt der Unterstrich `_` aus das wir den Parameter der Funktion ignorieren werden.
- Wir könnten nun `integrate(const_1)` aufrufen.
- Es gibt aber einen besseren, bequemeren Weg, Funktionen zu spezifizieren, die wir nur einmal verwenden wollen.
- Dieser Weg sind die sogenannten `lambda`s[51].
- `lambda`s sind namenlose Funktionen, die inline definiert werden.

```python
"""The syntax of a unary function always returning a constance value."""
def const_1(x: float) -> float:
    return 1.0  # always return 1.0, regardless of the value of x

# or

# It is common to use "_" as name for parameters that we will ignore.
def const_1(_: float) -> float:
    return 1.0  # always return 1.0, parameter "_" is ignored.
```

```python
"""The syntax of lambdas, i.e., inline function definitions."""

# This defines a nameless inline function with two parameters that
# returns "return_value".
lambda param1, param2: return_value
```

Implementierung

- Wir könnten nun `integrate(const_1)` aufrufen.
- Es gibt aber einen besseren, bequemeren Weg, Funktionen zu spezifizieren, die wir nur einmal verwenden wollen.
- Dieser Weg sind die sogenannten `lambda`s[51].
- `lambda`s sind namenlose Funktionen, die inline definiert werden.
- Sie fangen mit dem Schlüsselwort `lambda` an.

```python
"""The syntax of a unary function always returning a constance value."""
def const_1(x: float) -> float:
    return 1.0  # always return 1.0, regardless of the value of x

# or

# It is common to use "_" as name for parameters that we will ignore.
def const_1(_: float) -> float:
    return 1.0  # always return 1.0, parameter "_" is ignored.
```

```python
"""The syntax of lambdas, i.e., inline function definitions."""

# This defines a nameless inline function with two parameters that
# returns "return_value".
lambda param1, param2: return_value
```

Implementierung

- Es gibt aber einen besseren, bequemeren Weg, Funktionen zu spezifizieren, die wir nur einmal verwenden wollen.
- Dieser Weg sind die sogenannten `lambda`s[51].
- `lambda`s sind namenlose Funktionen, die inline definiert werden.
- Sie fangen mit dem Schlüsselwort `lambda` an.
- Dann kommen die Namen der Parameter, separiert mit Kommas `,`.

```
"""The syntax of a unary function always returning a constance value."""
def const_1(x: float) -> float:
    return 1.0  # always return 1.0, regardless of the value of x

# or

# It is common to use "_" as name for parameters that we will ignore.
def const_1(_: float) -> float:
    return 1.0  # always return 1.0, parameter "_" is ignored.
```

```
"""The syntax of lambdas, i.e., inline function definitions."""

# This defines a nameless inline function with two parameters that
# returns "return_value".
lambda param1, param2: return_value
```

Implementierung

- Dieser Weg sind die sogenannten `lambdas`[51].
- `lambdas` sind namenlose Funktionen, die inline definiert werden.
- Sie fangen mit dem Schlüsselwort `lambda` an.
- Dann kommen die Namen der Parameter, separiert mit Kommas `,`.
- Dann folgt ein Doppelpunkt `:` und danach kommt der Ausdruck, der den Rückgabewert der Inline-Funktion berechnet.

```python
"""The syntax of a unary function always returning a constance value."""
def const_1(x: float) -> float:
    return 1.0  # always return 1.0, regardless of the value of x

# or

# It is common to use "_" as name for parameters that we will ignore.
def const_1(_: float) -> float:
    return 1.0  # always return 1.0, parameter "_" is ignored.
```

```python
"""The syntax of lambdas, i.e., inline function definitions."""

# This defines a nameless inline function with two parameters that
# returns "return_value".
lambda param1, param2: return_value
```

Implementierung

- `lambda`s sind namenlose Funktionen, die inline definiert werden.
- Sie fangen mit dem Schlüsselwort `lambda` an.
- Dann kommen die Namen der Parameter, separiert mit Kommas `,`.
- Dann folgt ein Doppelpunkt `:` und danach kommt der Ausdruck, der den Rückgabewert der Inline-Funktion berechnet.
- Der Körper eines `lambda`s besteht nur aus diesem einzigen Ausdruck.

```python
"""The syntax of a unary function always returning a constance value."""
def const_1(x: float) -> float:
    return 1.0  # always return 1.0, regardless of the value of x

# or

# It is common to use "_" as name for parameters that we will ignore.
def const_1(_: float) -> float:
    return 1.0  # always return 1.0, parameter "_" is ignored.
```

```python
"""The syntax of lambdas, i.e., inline function definitions."""

# This defines a nameless inline function with two parameters that
# returns "return_value".
lambda param1, param2: return_value
```

Implementierung

- Sie fangen mit dem Schlüsselwort `lambda` an.
- Dann kommen die Namen der Parameter, separiert mit Kommas `,`.
- Dann folgt ein Doppelpunkt `:` und danach kommt der Ausdruck, der den Rückgabewert der Inline-Funktion berechnet.
- Der Körper eines `lambda`s besteht nur aus diesem einzigen Ausdruck.
- `lambda`s sind im Grunde eine einzelne Zeile Kode, die einen Rückgabewert berechnet.

```python
"""The syntax of a unary function always returning a constance value."""
def const_1(x: float) -> float:
    return 1.0  # always return 1.0, regardless of the value of x

# or

# It is common to use "_" as name for parameters that we will ignore.
def const_1(_: float) -> float:
    return 1.0  # always return 1.0, parameter "_" is ignored.
```

```python
"""The syntax of lambdas, i.e., inline function definitions."""

# This defines a nameless inline function with two parameters that
# returns "return_value".
lambda param1, param2: return_value
```

Implementierung

- Dann kommen die Namen der Parameter, separiert mit Kommas `,`.
- Dann folgt ein Doppelpunkt `:` und danach kommt der Ausdruck, der den Rückgabewert der Inline-Funktion berechnet.
- Der Körper eines `lambda`s besteht nur aus diesem einzigen Ausdruck.
- `lambda`s sind im Grunde eine einzelne Zeile Kode, die einen Rückgabewert berechnet.
- Wegen dem `:` in der Notation können wir sie auch nicht mit Type Hints annotieren.

```python
"""The syntax of a unary function always returning a constance value."""
def const_1(x: float) -> float:
    return 1.0  # always return 1.0, regardless of the value of x

# or

# It is common to use "_" as name for parameters that we will ignore.
def const_1(_: float) -> float:
    return 1.0  # always return 1.0, parameter "_" is ignored.
```

```python
"""The syntax of lambdas, i.e., inline function definitions."""

# This defines a nameless inline function with two parameters that
# returns "return_value".
lambda param1, param2: return_value
```

Implementierung

- Dann folgt ein Doppelpunkt `:` und danach kommt der Ausdruck, der den Rückgabewert der Inline-Funktion berechnet.
- Der Körper eines `lambda`s besteht nur aus diesem einzigen Ausdruck.
- `lambda`s sind im Grunde eine einzelne Zeile Kode, die einen Rückgabewert berechnet.
- Wegen dem `:` in der Notation können wir sie auch nicht mit Type Hints annotieren.
- Da `lambda`-Ausdrücke aber sehr klein sind und nur einmal verwendet werden, macht das nichts.

```
"""The syntax of a unary function always returning a constance value."""
def const_1(x: float) -> float:
    return 1.0  # always return 1.0, regardless of the value of x

# or

# It is common to use "_" as name for parameters that we will ignore.
def const_1(_: float) -> float:
    return 1.0  # always return 1.0, parameter "_" is ignored.
```

```
"""The syntax of lambdas, i.e., inline function definitions."""

# This defines a nameless inline function with two parameters that
# returns "return_value".
lambda param1, param2: return_value
```

Implementierung

- Der Körper eines `lambda`s besteht nur aus diesem einzigen Ausdruck.
- `lambda`s sind im Grunde eine einzelne Zeile Kode, die einen Rückgabewert berechnet.
- Wegen dem `:` in der Notation können wir sie auch nicht mit Type Hints annotieren.
- Da `lambda`-Ausdrücke aber sehr klein sind und nur einmal verwendet werden, macht das nichts.
- Wir wollen jetzt eine Funktion als `lambda` definieren, die die Konstante `1` zurückliefert.

```python
"""The syntax of a unary function always returning a constance value."""
def const_1(x: float) -> float:
    return 1.0  # always return 1.0, regardless of the value of x

# or

# It is common to use "_" as name for parameters that we will ignore.
def const_1(_: float) -> float:
    return 1.0  # always return 1.0, parameter "_" is ignored.
```

```python
"""The syntax of lambdas, i.e., inline function definitions."""

# This defines a nameless inline function with two parameters that
# returns "return_value".
lambda param1, param2: return_value
```

Implementierung

- `lambda`s sind im Grunde eine einzelne Zeile Kode, die einen Rückgabewert berechnet.
- Wegen dem `:` in der Notation können wir sie auch nicht mit Type Hints annotieren.
- Da `lambda`-Ausdrücke aber sehr klein sind und nur einmal verwendet werden, macht das nichts.
- Wir wollen jetzt eine Funktion als `lambda` definieren, die die Konstante `1` zurückliefert.
- Diese Funktion muss einen Parameter akzeptieren, anderfalls können wir sie nicht in `integrate` eingeben.

```python
"""The syntax of a unary function always returning a constance value."""
def const_1(x: float) -> float:
    return 1.0  # always return 1.0, regardless of the value of x

# or

# It is common to use "_" as name for parameters that we will ignore.
def const_1(_: float) -> float:
    return 1.0  # always return 1.0, parameter "_" is ignored.
```

```python
"""The syntax of lambdas, i.e., inline function definitions."""

# This defines a nameless inline function with two parameters that
# returns "return_value".
lambda param1, param2: return_value
```

Implementierung

- Wegen dem `:` in der Notation können wir sie auch nicht mit Type Hints annotieren.
- Da `lambda`-Ausdrücke aber sehr klein sind und nur einmal verwendet werden, macht das nichts.
- Wir wollen jetzt eine Funktion als `lambda` definieren, die die Konstante `1` zurückliefert.
- Diese Funktion muss einen Parameter akzeptieren, anderfalls können wir sie nicht in `integrate` eingeben.
- Da uns der Parameter egal ist, schreiben wir einfach folgendes:

```python
"""The syntax of a unary function always returning a constance value."""
def const_1(x: float) -> float:
    return 1.0  # always return 1.0, regardless of the value of x

# or

# It is common to use "_" as name for parameters that we will ignore.
def const_1(_: float) -> float:
    return 1.0  # always return 1.0, parameter "_" is ignored.
```

```python
"""The syntax of lambdas, i.e., inline function definitions."""

# This defines a nameless inline function with two parameters that
# returns "return_value".
lambda param1, param2: return_value
```

```python
"""lambdas can be used as values for Callable parameters."""

# If the Callable type hint indicates that the lambda should have a
# parameter, but we want to ignore that parameter, we should call it
# "_".
integrate(lambda _: 1.0)
```

Implementierung

- Da `lambda`-Ausdrücke aber sehr klein sind und nur einmal verwendet werden, macht das nichts.
- Wir wollen jetzt eine Funktion als `lambda` definieren, die die Konstante `1` zurückliefert.
- Diese Funktion muss einen Parameter akzeptieren, anderfalls können wir sie nicht in `integrate` eingeben.
- Da uns der Parameter egal ist, schreiben wir einfach folgendes:
- `lambda`s sind Funktionen, die wir nur einmal verwenden wollen.

```python
"""The syntax of a unary function always returning a constance value."""
def const_1(x: float) -> float:
    return 1.0  # always return 1.0, regardless of the value of x

# or

# It is common to use "_" as name for parameters that we will ignore.
def const_1(_: float) -> float:
    return 1.0  # always return 1.0, parameter "_" is ignored.
```

```python
"""The syntax of lambdas, i.e., inline function definitions."""

# This defines a nameless inline function with two parameters that
# returns "return_value".
lambda param1, param2: return_value
```

```python
"""lambdas can be used as values for Callable parameters."""

# If the Callable type hint indicates that the lambda should have a
# parameter, but we want to ignore that parameter, we should call it
# "_".
integrate(lambda _: 1.0)
```

Implementierung

- Wir wollen jetzt eine Funktion als `lambda` definieren, die die Konstante `1` zurückliefert.
- Diese Funktion muss einen Parameter akzeptieren, anderfalls können wir sie nicht in `integrate` eingeben.
- Da uns der Parameter egal ist, schreiben wir einfach folgendes:
- `lambda`s sind Funktionen, die wir nur einmal verwenden wollen.
- Das trifft klar auf unsere Funktion zu, die ihren Parameter ignoriert und immer `1.0` zurückliefert.

```python
"""The syntax of a unary function always returning a constance value."""
def const_1(x: float) -> float:
    return 1.0  # always return 1.0, regardless of the value of x

# or

# It is common to use "_" as name for parameters that we will ignore.
def const_1(_: float) -> float:
    return 1.0  # always return 1.0, parameter "_" is ignored.
```

```python
"""The syntax of lambdas, i.e., inline function definitions."""

# This defines a nameless inline function with two parameters that
# returns "return_value".
lambda param1, param2: return_value
```

```python
"""lambdas can be used as values for Callable parameters."""

# If the Callable type hint indicates that the lambda should have a
# parameter, but we want to ignore that parameter, we should call it
# "_".
integrate(lambda _: 1.0)
```

Implementierung

- Diese Funktion muss einen Parameter akzeptieren, anderfalls können wir sie nicht in `integrate` eingeben.
- Da uns der Parameter egal ist, schreiben wir einfach folgendes:
- `lambda`s sind Funktionen, die wir nur einmal verwenden wollen.
- Das trifft klar auf unsere Funktion zu, die ihren Parameter ignoriert und immer `1.0` zurückliefert.
- Wir können diesen Ausdruck also an `integrate` als Wert für Parameter `f` übergeben.

```python
"""The syntax of a unary function always returning a constance value."""
def const_1(x: float) -> float:
    return 1.0  # always return 1.0, regardless of the value of x

# or

# It is common to use "_" as name for parameters that we will ignore.
def const_1(_: float) -> float:
    return 1.0  # always return 1.0, parameter "_" is ignored.
```

```python
"""The syntax of lambdas, i.e., inline function definitions."""

# This defines a nameless inline function with two parameters that
# returns "return_value".
lambda param1, param2: return_value
```

```python
"""lambdas can be used as values for Callable parameters."""

# If the Callable type hint indicates that the lambda should have a
# parameter, but we want to ignore that parameter, we should call it
# "_".
integrate(lambda _: 1.0)
```

Implementierung

- Da uns der Parameter egal ist, schreiben wir einfach folgendes:
- `lambda`s sind Funktionen, die wir nur einmal verwenden wollen.
- Das trifft klar auf unsere Funktion zu, die ihren Parameter ignoriert und immer `1.0` zurückliefert.
- Wir können diesen Ausdruck also an `integrate` als Wert für Parameter `f` übergeben.
- Natürlich ist die Fläche unter der konstanten Funktion $f(x) = 1$ über dem Intervall $[0, 1]$ auch 1.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

↓ `python3 integral.py` ↓

```
∫1dx|0,1 ≈ 1.0
∫x²-2dx|-1,1 ≈ -3.3332
∫sin(x)dx|0,π ≈ 1.9999588764792162
∫f(x,0,1)dx|-1,1 ≈ 0.6826733605403601
∫f(x,0,1)dx|-2,2 ≈ 0.9544709416896361
```

Implementierung

- `lambda`s sind Funktionen, die wir nur einmal verwenden wollen.
- Das trifft klar auf unsere Funktion zu, die ihren Parameter ignoriert und immer `1.0` zurückliefert.
- Wir können diesen Ausdruck also an `integrate` als Wert für Parameter `f` übergeben.
- Natürlich ist die Fläche unter der konstanten Funktion $f(x) = 1$ über dem Intervall $[0, 1]$ auch 1.
- Mit `n=7` Trapezen bekommen wir genau dieses Ergebnis.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

↓ `python3 integral.py` ↓

```
∫1dx|0,1 ≈ 1.0
∫x²-2dx|-1,1 ≈ -3.3332
∫sin(x)dx|0,π ≈ 1.9999588764792162
∫f(x,0,1)dx|-1,1 ≈ 0.6826733605403601
∫f(x,0,1)dx|-2,2 ≈ 0.9544709416896361
```

Implementierung

- Das trifft klar auf unsere Funktion zu, die ihren Parameter ignoriert und immer `1.0` zurückliefert.
- Wir können diesen Ausdruck also an `integrate` als Wert für Parameter `f` übergeben.
- Natürlich ist die Fläche unter der konstanten Funktion $f(x) = 1$ über dem Intervall $[0, 1]$ auch 1.
- Mit `n=7` Trapezen bekommen wir genau dieses Ergebnis.
- Unsere Funktion `integrate` hat also den ersten Test bestanden.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

↓ `python3 integral.py` ↓

```
∫1dx|0,1 ≈ 1.0
∫x²-2dx|-1,1 ≈ -3.3332
∫sin(x)dx|0,π ≈ 1.9999588764792162
∫f(x,0,1)dx|-1,1 ≈ 0.6826733605403601
∫f(x,0,1)dx|-2,2 ≈ 0.9544709416896361
```

Implementierung

- Wir können diesen Ausdruck also an `integrate` als Wert für Parameter `f` übergeben.
- Natürlich ist die Fläche unter der konstanten Funktion $f(x) = 1$ über dem Intervall $[0, 1]$ auch 1.
- Mit `n=7` Trapezen bekommen wir genau dieses Ergebnis.
- Unsere Funktion `integrate` hat also den ersten Test bestanden.
- In der Ausgabe unseres Programmes schreiben wir $\int 1 dx |_{0,1}$ wobei das `|0,1` die Intervallgrenzen $[0, 1]$ beschreibt.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

↓ `python3 integral.py` ↓

```
∫1dx|0,1 ≈ 1.0
∫x²-2dx|-1,1 ≈ -3.3332
∫sin(x)dx|0,π ≈ 1.9999588764792162
∫f(x,0,1)dx|-1,1 ≈ 0.6826733605403601
∫f(x,0,1)dx|-2,2 ≈ 0.9544709416896361
```

Implementierung

- Natürlich ist die Fläche unter der konstanten Funktion $f(x) = 1$ über dem Intervall $[0, 1]$ auch 1.
- Mit `n=7` Trapezen bekommen wir genau dieses Ergebnis.
- Unsere Funktion `integrate` hat also den ersten Test bestanden.
- In der Ausgabe unseres Programmes schreiben wir `∫1dx|0,1` wobei das `|0,1` die Intervallgrenzen $[0, 1]$ beschreibt.
- Wir benutzen den Unicode-Escape `"\u222b"` um das $\int$-Zeichen in dem f-String darzustellen.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

↓ `python3 integral.py` ↓

```
∫1dx|0,1 ≈ 1.0
∫x²-2dx|-1,1 ≈ -3.3332
∫sin(x)dx|0,π ≈ 1.9999588764792162
∫f(x,0,1)dx|-1,1 ≈ 0.6826733605403601
∫f(x,0,1)dx|-2,2 ≈ 0.9544709416896361
```

Implementierung

- Mit `n=7` Trapezen bekommen wir genau dieses Ergebnis.
- Unsere Funktion `integrate` hat also den ersten Test bestanden.
- In der Ausgabe unseres Programmes schreiben wir `∫1dx|0,1` wobei das `|0,1` die Intervallgrenzen $[0, 1]$ beschreibt.
- Wir benutzen den Unicode-Escape `"\u222b"` um das $\int$-Zeichen in dem f-String darzustellen.
- Versuchen wir nun die Fläche unter der Funktion $g(x) = x^2 - 2$ über dem Intervall $[-1, 1]$, also $\int_{-1}^{1} x^2 - 2\,dx$.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

↓ `python3 integral.py` ↓

```
∫1dx|0,1 ≈ 1.0
∫x²-2dx|-1,1 ≈ -3.3332
∫sin(x)dx|0,π ≈ 1.9999588764792162
∫f(x,0,1)dx|-1,1 ≈ 0.6826733605403601
∫f(x,0,1)dx|-2,2 ≈ 0.9544709416896361
```

Implementierung

- In der Ausgabe unseres Programmes schreiben wir `∫1dx|0,1` wobei das `|0,1` die Intervallgrenzen $[0, 1]$ beschreibt.
- Wir benutzen den Unicode-Escape `"\u222b"` um das $\int$-Zeichen in dem f-String darzustellen.
- Versuchen wir nun die Fläche unter der Funktion $g(x) = x^2 - 2$ über dem Intervall $[-1, 1]$, also $\int_{-1}^{1} x^2 - 2\,dx$.
- Die Stammfunktion von $g(x)$ ist $G(x) = \frac{1}{3}x^3 - 2x + c$ und $G(1) - G(-1) = [\frac{1}{3} - 2] - [-\frac{1}{3} + 2] = \frac{2}{3} - 4 = -3\frac{1}{3} = -3.\overline{3}$.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

↓ `python3 integral.py` ↓

```
∫1dx|0,1 ≈ 1.0
∫x²-2dx|-1,1 ≈ -3.3332
∫sin(x)dx|0,π ≈ 1.9999588764792162
∫f(x,0,1)dx|-1,1 ≈ 0.6826733605403601
∫f(x,0,1)dx|-2,2 ≈ 0.9544709416896361
```

Implementierung

- Wir benutzen den Unicode-Escape `"\u222b"` um das $\int$-Zeichen in dem f-String darzustellen.
- Versuchen wir nun die Fläche unter der Funktion $g(x) = x^2 - 2$ über dem Intervall $[-1, 1]$, also $\int_{-1}^{1} x^2 - 2\,dx$.
- Die Stammfunktion von $g(x)$ ist $G(x) = \frac{1}{3}x^3 - 2x + c$ und $G(1) - G(-1) = [\frac{1}{3} - 2] - [-\frac{1}{3} + 2] = \frac{2}{3} - 4 = -3\frac{1}{3} = -3.\overline{3}$.
- Wir geben die Funktion $g(x)$ als `lambda`-Ausdruck in unsere `integrate`-Funktion in dem wir schreiben `lambda x: x*x - 2`.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx\u00b2-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

↓ `python3 integral.py` ↓

```
∫1dx|0,1 ≈ 1.0
∫x²-2dx|-1,1 ≈ -3.3332
∫sin(x)dx|0,π ≈ 1.9999588764792162
∫f(x,0,1)dx|-1,1 ≈ 0.6826733605403601
∫f(x,0,1)dx|-2,2 ≈ 0.9544709416896361
```

Implementierung

- Versuchen wir nun die Fläche unter der Funktion $g(x) = x^2 - 2$ über dem Intervall $[-1, 1]$, also $\int_{-1}^{1} x^2 - 2\,dx$.
- Die Stammfunktion von $g(x)$ ist $G(x) = \frac{1}{3}x^3 - 2x + c$ und $G(1) - G(-1) = [\frac{1}{3} - 2] - [-\frac{1}{3} + 2] = \frac{2}{3} - 4 = -3\frac{1}{3} = -3.\overline{3}$.
- Wir geben die Funktion $g(x)$ als `lambda`-Ausdruck in unsere `integrate`-Funktion in dem wir schreiben `lambda x: x*x - 2`.
- Wir müssen auch den Wert für `a` angeben, nämlich `-1`.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

↓ `python3 integral.py` ↓

```
∫1dx|0,1 ≈ 1.0
∫x²-2dx|-1,1 ≈ -3.3332
∫sin(x)dx|0,π ≈ 1.9999588764792162
∫f(x,0,1)dx|-1,1 ≈ 0.6826733605403601
∫f(x,0,1)dx|-2,2 ≈ 0.9544709416896361
```

Implementierung

- Die Stammfunktion von $g(x)$ ist $G(x) = \frac{1}{3}x^3 - 2x + c$ und $G(1) - G(-1) = [\frac{1}{3} - 2] - [-\frac{1}{3} + 2] = \frac{2}{3} - 4 = -3\frac{1}{3} = -3.\overline{3}$.
- Wir geben die Funktion $g(x)$ als `lambda`-Ausdruck in unsere `integrate`-Funktion in dem wir schreiben `lambda x: x*x - 2`.
- Wir müssen auch den Wert für `a` angeben, nämlich `-1`.
- Mit dem Default Value von `n`=100 Schritten liefert unsere Funktion `-3.3332`, was schon sehr nahe an $-3.\overline{3}$ ist.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

↓ `python3 integral.py` ↓

```
∫1dx|0,1 ≈ 1.0
∫x²-2dx|-1,1 ≈ -3.3332
∫sin(x)dx|0,π ≈ 1.9999588764792162
∫f(x,0,1)dx|-1,1 ≈ 0.6826733605403601
∫f(x,0,1)dx|-2,2 ≈ 0.9544709416896361
```

Implementierung

- Wir geben die Funktion $g(x)$ als `lambda`-Ausdruck in unsere `integrate`-Funktion in dem wir schreiben `lambda x: x*x - 2`.
- Wir müssen auch den Wert für `a` angeben, nämlich `-1`.
- Mit dem Default Value von `n`=100 Schritten liefert unsere Funktion `-3.3332`, was schon sehr nahe an $-3.\overline{3}$ ist.
- Integrieren wir nun die Sinusfunktion über dem Intervall $[0, \pi]$.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

↓ `python3 integral.py` ↓

```
∫1dx|0,1 ≈ 1.0
∫x²-2dx|-1,1 ≈ -3.3332
∫sin(x)dx|0,π ≈ 1.9999588764792162
∫f(x,0,1)dx|-1,1 ≈ 0.6826733605403601
∫f(x,0,1)dx|-2,2 ≈ 0.9544709416896361
```

Implementierung

- Wir geben die Funktion $g(x)$ als `lambda`-Ausdruck in unsere `integrate`-Funktion in dem wir schreiben `lambda x: x*x - 2`.
- Wir müssen auch den Wert für `a` angeben, nämlich `-1`.
- Mit dem Default Value von `n`=100 Schritten liefert unsere Funktion `-3.3332`, was schon sehr nahe an $-3.\overline{3}$ ist.
- Integrieren wir nun die Sinusfunktion über dem Intervall $[0, \pi]$.
- Dazu importieren wir die Funktion `sin` aus dem Modul `math` und geben sie als Argument `f` an `integrate`.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

↓ `python3 integral.py` ↓

```
∫1dx|0,1 ≈ 1.0
∫x²-2dx|-1,1 ≈ -3.3332
∫sin(x)dx|0,π ≈ 1.9999588764792162
∫f(x,0,1)dx|-1,1 ≈ 0.6826733605403601
∫f(x,0,1)dx|-2,2 ≈ 0.9544709416896361
```

Implementierung

- Wir müssen auch den Wert für `a` angeben, nämlich `-1`.
- Mit dem Default Value von `n=100` Schritten liefert unsere Funktion `-3.3332`, was schon sehr nahe an $-3.\overline{3}$ ist.
- Integrieren wir nun die Sinusfunktion über dem Intervall $[0, \pi]$.
- Dazu importieren wir die Funktion `sin` aus dem Modul `math` und geben sie als Argument `f` an `integrate`.
- Wir müssen auch `b = pi` angeben, wobei wir `pi` auch aus `math` importieren.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

↓ `python3 integral.py` ↓

```
∫1dx|0,1 ≈ 1.0
∫x²-2dx|-1,1 ≈ -3.3332
∫sin(x)dx|0,π ≈ 1.9999588764792162
∫f(x,0,1)dx|-1,1 ≈ 0.6826733605403601
∫f(x,0,1)dx|-2,2 ≈ 0.9544709416896361
```

Implementierung

- Mit dem Default Value von `n=100` Schritten liefert unsere Funktion `-3.3332`, was schon sehr nahe an $-3.\overline{3}$ ist.
- Integrieren wir nun die Sinusfunktion über dem Intervall $[0, \pi]$.
- Dazu importieren wir die Funktion `sin` aus dem Modul `math` und geben sie als Argument `f` an `integrate`.
- Wir müssen auch `b = pi` angeben, wobei wir `pi` auch aus `math` importieren.
- Dieses mal benutzen wir `n = 200` Trapeze.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

↓ `python3 integral.py` ↓

```
∫1dx|0,1 ≈ 1.0
∫x²-2dx|-1,1 ≈ -3.3332
∫sin(x)dx|0,π ≈ 1.9999588764792162
∫f(x,0,1)dx|-1,1 ≈ 0.6826733605403601
∫f(x,0,1)dx|-2,2 ≈ 0.9544709416896361
```

Implementierung

- Integrieren wir nun die Sinusfunktion über dem Intervall $[0, \pi]$.
- Dazu importieren wir die Funktion `sin` aus dem Modul `math` und geben sie als Argument `f` an `integrate`.
- Wir müssen auch `b = pi` angeben, wobei wir `pi` auch aus `math` importieren.
- Dieses mal benutzen wir `n = 200` Trapeze.
- Die Stammfunktion von $\sin x$ ist $-\cos x + c$, wodurch das erwartete Ergebnis $[-\cos \pi] - [-\cos 0] = [-(-1)] - [-1] = 2$ ist.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

↓ `python3 integral.py` ↓

```
∫1dx|0,1 ≈ 1.0
∫x²-2dx|-1,1 ≈ -3.3332
∫sin(x)dx|0,π ≈ 1.9999588764792162
∫f(x,0,1)dx|-1,1 ≈ 0.6826733605403601
∫f(x,0,1)dx|-2,2 ≈ 0.9544709416896361
```

Implementierung

- Dazu importieren wir die Funktion `sin` aus dem Modul `math` und geben sie als Argument `f` an `integrate`.
- Wir müssen auch `b = pi` angeben, wobei wir `pi` auch aus `math` importieren.
- Dieses mal benutzen wir `n = 200` Trapeze.
- Die Stammfunktion von $\sin x$ ist $-\cos x + c$, wodurch das erwartete Ergebnis $[-\cos \pi] - [-\cos 0] = [-(-1)] - [-1] = 2$ ist.
- Unsere Funktion liefert uns 1.99996, was wiederum sehr nahe dran ist.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

↓ `python3 integral.py` ↓

```
∫1dx|0,1 ≈ 1.0
∫x²-2dx|-1,1 ≈ -3.3332
∫sin(x)dx|0,π ≈ 1.9999588764792162
∫f(x,0,1)dx|-1,1 ≈ 0.6826733605403601
∫f(x,0,1)dx|-2,2 ≈ 0.9544709416896361
```

Implementierung

- Wir müssen auch `b = pi` angeben, wobei wir `pi` auch aus `math` importieren.
- Dieses mal benutzen wir `n = 200` Trapeze.
- Die Stammfunktion von $\sin x$ ist $-\cos x + c$, wodurch das erwartete Ergebnis $[-\cos \pi] - [-\cos 0] = [-(-1)] - [-1] = 2$ ist.
- Unsere Funktion liefert uns 1.99996, was wiederum sehr nahe dran ist.
- Als letztes Beispiel integrieren wir die PDF der Normalverteilung.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

↓ `python3 integral.py` ↓

```
∫1dx|0,1 ≈ 1.0
∫x²-2dx|-1,1 ≈ -3.3332
∫sin(x)dx|0,π ≈ 1.9999588764792162
∫f(x,0,1)dx|-1,1 ≈ 0.6826733605403601
∫f(x,0,1)dx|-2,2 ≈ 0.9544709416896361
```

Implementierung

- Dieses mal benutzen wir `n = 200` Trapeze.
- Die Stammfunktion von $\sin x$ ist $-\cos x + c$, wodurch das erwartete Ergebnis $[-\cos \pi] - [-\cos 0] = [-(-1)] - [-1] = 2$ ist.
- Unsere Funktion liefert uns 1.99996, was wiederum sehr nahe dran ist.
- Als letztes Beispiel integrieren wir die PDF der Normalverteilung.
- Wir haben diese ja schon in der letzten Einheit implementiert.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

↓ `python3 integral.py` ↓

```
∫1dx|0,1 ≈ 1.0
∫x²-2dx|-1,1 ≈ -3.3332
∫sin(x)dx|0,π ≈ 1.9999588764792162
∫f(x,0,1)dx|-1,1 ≈ 0.6826733605403601
∫f(x,0,1)dx|-2,2 ≈ 0.9544709416896361
```

Implementierung

- Die Stammfunktion von $\sin x$ ist $-\cos x + c$, wodurch das erwartete Ergebnis $[-\cos \pi] - [-\cos 0] = [-(-1)] - [-1] = 2$ ist.
- Unsere Funktion liefert uns 1.99996, was wiederum sehr nahe dran ist.
- Als letztes Beispiel integrieren wir die PDF der Normalverteilung.
- Wir haben diese ja schon in der letzten Einheit implementiert.
- Diese Funktion hat aber genau genommen drei Parameter, nämlich `x`, `mu`, und `sigma`.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

↓ `python3 integral.py` ↓

```
∫1dx|0,1 ≈ 1.0
∫x²-2dx|-1,1 ≈ -3.3332
∫sin(x)dx|0,π ≈ 1.9999588764792162
∫f(x,0,1)dx|-1,1 ≈ 0.6826733605403601
∫f(x,0,1)dx|-2,2 ≈ 0.9544709416896361
```

Implementierung

- Unsere Funktion liefert uns 1.99996, was wiederum sehr nahe dran ist.
- Als letztes Beispiel integrieren wir die PDF der Normalverteilung.
- Wir haben diese ja schon in der letzten Einheit implementiert.
- Diese Funktion hat aber genau genommen drei Parameter, nämlich `x`, `mu`, und `sigma`.
- Die letzten beiden haben die Default Values `mu = 0.0` und `sigma = 1.0`.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

↓ `python3 integral.py` ↓

```
∫1dx|0,1 ≈ 1.0
∫x²-2dx|-1,1 ≈ -3.3332
∫sin(x)dx|0,π ≈ 1.9999588764792162
∫f(x,0,1)dx|-1,1 ≈ 0.6826733605403601
∫f(x,0,1)dx|-2,2 ≈ 0.9544709416896361
```

Implementierung

- Als letztes Beispiel integrieren wir die PDF der Normalverteilung.
- Wir haben diese ja schon in der letzten Einheit implementiert.
- Diese Funktion hat aber genau genommen drei Parameter, nämlich `x`, `mu`, und `sigma`.
- Die letzten beiden haben die Default Values `mu = 0.0` und `sigma = 1.0`.
- Wenn diese Default Values benutzt werden, berechnet `pdf` die PDF die Werte der Standardnormalverteilung.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

↓ `python3 integral.py` ↓

```
∫1dx|0,1 ≈ 1.0
∫x²-2dx|-1,1 ≈ -3.3332
∫sin(x)dx|0,π ≈ 1.9999588764792162
∫f(x,0,1)dx|-1,1 ≈ 0.6826733605403601
∫f(x,0,1)dx|-2,2 ≈ 0.9544709416896361
```

Implementierung

- Wir haben diese ja schon in der letzten Einheit implementiert.
- Diese Funktion hat aber genau genommen drei Parameter, nämlich `x`, `mu`, und `sigma`.
- Die letzten beiden haben die Default Values `mu = 0.0` und `sigma = 1.0`.
- Wenn diese Default Values benutzt werden, berechnet `pdf` die PDF die Werte der Standardnormalverteilung.
- Interessanterweise, obwohl die Funktion `pdf` drei Parameter hat und der Funktionsparameter `f` nur einen erwartet, können wir `pdf` trotzdem als `f` eingeben.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

↓ `python3 integral.py` ↓

```
∫1dx|0,1 ≈ 1.0
∫x²-2dx|-1,1 ≈ -3.3332
∫sin(x)dx|0,π ≈ 1.9999588764792162
∫f(x,0,1)dx|-1,1 ≈ 0.6826733605403601
∫f(x,0,1)dx|-2,2 ≈ 0.9544709416896361
```

Implementierung

- Diese Funktion hat aber genau genommen drei Parameter, nämlich `x`, `mu`, und `sigma`.
- Die letzten beiden haben die Default Values `mu = 0.0` und `sigma = 1.0`.
- Wenn diese Default Values benutzt werden, berechnet `pdf` die PDF die Werte der Standardnormalverteilung.
- Interessanterweise, obwohl die Funktion `pdf` drei Parameter hat und der Funktionsparameter `f` nur einen erwartet, können wir `pdf` trotzdem als `f` eingeben.
- Das geht genau deshalb, weil der zweite und dritte Parameter Default Values haben.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

↓ `python3 integral.py` ↓

```
∫1dx|0,1 ≈ 1.0
∫x²-2dx|-1,1 ≈ -3.3332
∫sin(x)dx|0,π ≈ 1.9999588764792162
∫f(x,0,1)dx|-1,1 ≈ 0.6826733605403601
∫f(x,0,1)dx|-2,2 ≈ 0.9544709416896361
```

Implementierung

- Die letzten beiden haben die Default Values `mu = 0.0` und `sigma = 1.0`.
- Wenn diese Default Values benutzt werden, berechnet `pdf` die PDF die Werte der Standardnormalverteilung.
- Interessanterweise, obwohl die Funktion `pdf` drei Parameter hat und der Funktionsparameter `f` nur einen erwartet, können wir `pdf` trotzdem als `f` eingeben.
- Das geht genau deshalb, weil der zweite und dritte Parameter Default Values haben.
- Selbst Mypy acceptiert das.

```
$ mypy integral.py --no-strict-optional --check-untyped-defs
Success: no issues found in 1 source file
# mypy 1.19.1 succeeded with exit code 0.
```

Implementierung

- Wenn diese Default Values benutzt werden, berechnet `pdf` die PDF die Werte der Standardnormalverteilung.
- Interessanterweise, obwohl die Funktion `pdf` drei Parameter hat und der Funktionsparameter `f` nur einen erwartet, können wir `pdf` trotzdem als `f` eingeben.
- Das geht genau deshalb, weil der zweite und dritte Parameter Default Values haben.
- Selbst Mypy acceptiert das.
- Die Standardnormalverteilung hat Standardabweichung $\sigma = 1$ und Mittelwert $\mu = 0$.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

↓ `python3 integral.py` ↓

```
∫1dx|0,1 ≈ 1.0
∫x²-2dx|-1,1 ≈ -3.3332
∫sin(x)dx|0,π ≈ 1.9999588764792162
∫f(x,0,1)dx|-1,1 ≈ 0.6826733605403601
∫f(x,0,1)dx|-2,2 ≈ 0.9544709416896361
```

Implementierung

- Das geht genau deshalb, weil der zweite und dritte Parameter Default Values haben.
- Selbst Mypy acceptiert das.
- Die Standardnormalverteilung hat Standardabweichung $\sigma = 1$ und Mittelwert $\mu = 0$.
- Wenn wir über das Intervall $[-1, 1]$ integrieren, dann berechnen wir die Wahrscheinlichkeit, dass eine standardnormalverteilte Zufallsvariable $\mathcal{X}$ aus dem Intervall $[\mu - \sigma, \mu + \sigma]$ gezogen wird, also innerhalb einer Standardabweichung vom Mittelwert.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

↓ `python3 integral.py` ↓

```
∫1dx|0,1 ≈ 1.0
∫x²-2dx|-1,1 ≈ -3.3332
∫sin(x)dx|0,π ≈ 1.9999588764792162
∫f(x,0,1)dx|-1,1 ≈ 0.6826733605403601
∫f(x,0,1)dx|-2,2 ≈ 0.9544709416896361
```

Implementierung

- Selbst Mypy acceptiert das.
- Die Standardnormalverteilung hat Standardabweichung $\sigma = 1$ und Mittelwert $\mu = 0$.
- Wenn wir über das Intervall $[-1, 1]$ integrieren, dann berechnen wir die Wahrscheinlichkeit, dass eine standardnormalverteilte Zufallsvariable $\mathcal{X}$ aus dem Intervall $[\mu - \sigma, \mu + \sigma]$ gezogen wird, also innerhalb einer Standardabweichung vom Mittelwert.
- Wie Sie vielleicht noch aus der Schule wissen, ist diese Wahrscheinlichkeit etwa 68.26%[1,32,98].

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

↓ `python3 integral.py` ↓

```
∫1dx|0,1 ≈ 1.0
∫x²-2dx|-1,1 ≈ -3.3332
∫sin(x)dx|0,π ≈ 1.9999588764792162
∫f(x,0,1)dx|-1,1 ≈ 0.6826733605403601
∫f(x,0,1)dx|-2,2 ≈ 0.9544709416896361
```

Implementierung

- Wenn wir über das Intervall $[-1, 1]$ integrieren, dann berechnen wir die Wahrscheinlichkeit, dass eine standardnormalverteilte Zufallsvariable $\mathcal{X}$ aus dem Intervall $[\mu - \sigma, \mu + \sigma]$ gezogen wird, also innerhalb einer Standardabweichung vom Mittelwert.
- Wie Sie vielleicht noch aus der Schule wissen, ist diese Wahrscheinlichkeit etwa 68.26%[1,32,98].
- Die Wahrscheinlichkeit, die Variable aus $[-2, 2]$ zu ziehen, also nicht mehr als zwei Standardabweichungen vom Mittel, ist etwa 95.44%[1,32,98].

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

↓ `python3 integral.py` ↓

```
∫1dx|0,1 ≈ 1.0
∫x²-2dx|-1,1 ≈ -3.3332
∫sin(x)dx|0,π ≈ 1.9999588764792162
∫f(x,0,1)dx|-1,1 ≈ 0.6826733605403601
∫f(x,0,1)dx|-2,2 ≈ 0.9544709416896361
```

Implementierung

- Wie Sie vielleicht noch aus der Schule wissen, ist diese Wahrscheinlichkeit etwa 68.26%[1,32,98].
- Die Wahrscheinlichkeit, die Variable aus $[-2, 2]$ zu ziehen, also nicht mehr als zwei Standardabweichungen vom Mittel, ist etwa 95.44%[1,32,98].
- Unsere kleine Funktion berechnet das schon ganz gut.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

↓ `python3 integral.py` ↓

```
∫1dx|0,1 ≈ 1.0
∫x²-2dx|-1,1 ≈ -3.3332
∫sin(x)dx|0,π ≈ 1.9999588764792162
∫f(x,0,1)dx|-1,1 ≈ 0.6826733605403601
∫f(x,0,1)dx|-2,2 ≈ 0.9544709416896361
```

Implementierung

- Wie Sie vielleicht noch aus der Schule wissen, ist diese Wahrscheinlichkeit etwa 68.26%[1,32,98].
- Die Wahrscheinlichkeit, die Variable aus $[-2, 2]$ zu ziehen, also nicht mehr als zwei Standardabweichungen vom Mittel, ist etwa 95.44%[1,32,98].
- Unsere kleine Funktion berechnet das schon ganz gut.
- Zuletzt sei noch darauf hingewiesen, dass die Trapez-basierte Annäherung bestimmter Integrale nicht unbedingt die beste Methode ist.

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

↓ `python3 integral.py` ↓

```
∫1dx|0,1 ≈ 1.0
∫x²-2dx|-1,1 ≈ -3.3332
∫sin(x)dx|0,π ≈ 1.9999588764792162
∫f(x,0,1)dx|-1,1 ≈ 0.6826733605403601
∫f(x,0,1)dx|-2,2 ≈ 0.9544709416896361
```

Implementierung

- Die Wahrscheinlichkeit, die Variable aus $[-2, 2]$ zu ziehen, also nicht mehr als zwei Standardabweichungen vom Mittel, ist etwa 95.44%[1,32,98].
- Unsere kleine Funktion berechnet das schon ganz gut.
- Zuletzt sei noch darauf hingewiesen, dass die Trapez-basierte Annäherung bestimmter Integrale nicht unbedingt die beste Methode ist.
- Andere Methoden, z. B. Simpson's Regel, können oft bessere Ergebnisse liefern[28].

```python
"""Numerical integration using the trapezoid method."""

from math import pi, sin
from typing import Callable

from normal_pdf import pdf


def integrate(
        f: Callable[[float], float | int], a: int | float = 0.0,
        b: int | float = 1.0, n: int = 100) -> float:
    """
    Integrate the function `f` between `a` and `b` using trapezoids.

    :param f: the function to integrate: in float, out float or int
    :param a: the lower end of the range over which to integrate
    :param b: the upper end of the range over which to integrate
    :param n: the number of trapezoids to use for the approximation
    :return: the approximated integration result
    """
    result: float = 0.5 * (f(a) + f(b))  # Initialize with start + end.
    h: float = (b - a) / n  # The base length of the trapezoids.
    for i in range(1, n):  # The steps between start and end.
        result += f(a + h * i)  # Add f(x) between trapezoids.
    return result * h  # Multiply result with base length.


print(f"\u222b1dx|0,1 \u2248 {integrate(lambda _: 1, n=7)}")
print(f"\u222bx²-2dx|-1,1 \u2248 {integrate(lambda x: x * x - 2, -1)}")
print(f"\u222bsin(x)dx|0,\u03C0 \u2248 {integrate(sin, b=pi, n=200)}")
print(f"\u222bf(x,0,1)dx|-1,1 \u2248 {integrate(pdf, -1)}")
print(f"\u222bf(x,0,1)dx|-2,2 \u2248 {integrate(pdf, -2, 2)}")
```

↓ `python3 integral.py` ↓

```
∫1dx|0,1 ≈ 1.0
∫x²-2dx|-1,1 ≈ -3.3332
∫sin(x)dx|0,π ≈ 1.9999588764792162
∫f(x,0,1)dx|-1,1 ≈ 0.6826733605403601
∫f(x,0,1)dx|-2,2 ≈ 0.9544709416896361
```

Zusammenfassung

Funktionen

- Funktionen sind der zentrale Baustein für modularen Kode.

Funktionen

- Funktionen sind der zentrale Baustein für modularen Kode.
- Sie erlauben es uns, Kode in Einheiten mit klar strukturierten Schnittstellen zu gruppieren.

Funktionen

- Funktionen sind der zentrale Baustein für modularen Kode.
- Sie erlauben es uns, Kode in Einheiten mit klar strukturierten Schnittstellen zu gruppieren.
- Funktionen können Eingabedaten über Parameter erhalten.

Funktionen

- Funktionen sind der zentrale Baustein für modularen Kode.
- Sie erlauben es uns, Kode in Einheiten mit klar strukturierten Schnittstellen zu gruppieren.
- Funktionen können Eingabedaten über Parameter erhalten.
- Sie können Ergebnisse als Rückgabewert zurückliefern.

Funktionen

- Funktionen sind der zentrale Baustein für modularen Kode.
- Sie erlauben es uns, Kode in Einheiten mit klar strukturierten Schnittstellen zu gruppieren.
- Funktionen können Eingabedaten über Parameter erhalten.
- Sie können Ergebnisse als Rückgabewert zurückliefern.
- Sowohl Parameter als auch Rückgabewerte können über Type Hints annotiert werden.

Funktionen

- Funktionen sind der zentrale Baustein für modularen Kode.
- Sie erlauben es uns, Kode in Einheiten mit klar strukturierten Schnittstellen zu gruppieren.
- Funktionen können Eingabedaten über Parameter erhalten.
- Sie können Ergebnisse als Rückgabewert zurückliefern.
- Sowohl Parameter als auch Rückgabewerte können über Type Hints annotiert werden.
- Die Beschreibung, was eine Funktion macht und was Parameter und Rückgabewerte bedeuten kommt in den Docstring.

Funktionen: Vorteile

Funktionen haben viele Vorteile.

Funktionen: Vorteile

Funktionen haben viele Vorteile.

1. Mehrere Programmierer können gemeinsam an einem Projekt arbeiten.

Funktionen: Vorteile

Funktionen haben viele Vorteile.

1. Mehrere Programmierer können gemeinsam an einem Projekt arbeiten. Sie können an verschiedenen Funktionen arbeiten und diese in verschiedenen Modulen gruppieren.

Funktionen: Vorteile

Funktionen haben viele Vorteile.

1. Mehrere Programmierer können gemeinsam an einem Projekt arbeiten. Sie können an verschiedenen Funktionen arbeiten und diese in verschiedenen Modulen gruppieren.
2. Durch Type Hints und Docstrings sind Funktionen leicht für andere Programmierer zu verstehen.

Funktionen: Vorteile

Funktionen haben viele Vorteile.

1. Mehrere Programmierer können gemeinsam an einem Projekt arbeiten. Sie können an verschiedenen Funktionen arbeiten und diese in verschiedenen Modulen gruppieren.
2. Durch Type Hints und Docstrings sind Funktionen leicht für andere Programmierer zu verstehen. Werkzeuge wie Mypy können prüfen, ob die übergebenen Werte die richtigen Typen haben.

Funktionen: Vorteile

Funktionen haben viele Vorteile.

1. Mehrere Programmierer können gemeinsam an einem Projekt arbeiten. Sie können an verschiedenen Funktionen arbeiten und diese in verschiedenen Modulen gruppieren.
2. Durch Type Hints und Docstrings sind Funktionen leicht für andere Programmierer zu verstehen. Werkzeuge wie Mypy können prüfen, ob die übergebenen Werte die richtigen Typen haben.
3. Funktionen erlauben es uns, Kode an verschiedenen Stellen wiederzuverwenden.

Funktionen: Vorteile

Funktionen haben viele Vorteile.

1. Mehrere Programmierer können gemeinsam an einem Projekt arbeiten. Sie können an verschiedenen Funktionen arbeiten und diese in verschiedenen Modulen gruppieren.
2. Durch Type Hints und Docstrings sind Funktionen leicht für andere Programmierer zu verstehen. Werkzeuge wie Mypy können prüfen, ob die übergebenen Werte die richtigen Typen haben.
3. Funktionen erlauben es uns, Kode an verschiedenen Stellen wiederzuverwenden. Wenn wir die Fläche unter anderen Funktionen berechnen wollen, können wir unsere `integrate`-Funktion immer wiederverwenden.

Funktionen: Vorteile

Funktionen haben viele Vorteile.

1. Mehrere Programmierer können gemeinsam an einem Projekt arbeiten. Sie können an verschiedenen Funktionen arbeiten und diese in verschiedenen Modulen gruppieren.
2. Durch Type Hints und Docstrings sind Funktionen leicht für andere Programmierer zu verstehen. Werkzeuge wie Mypy können prüfen, ob die übergebenen Werte die richtigen Typen haben.
3. Funktionen erlauben es uns, Kode an verschiedenen Stellen wiederzuverwenden. Wenn wir die Fläche unter anderen Funktionen berechnen wollen, können wir unsere `integrate`-Funktion immer wiederverwenden.
4. Wenn wir unseren Kode in Funktionen mit klaren Eingabe- und Ausgabedaten strukturieren, dann können wir diese einzeln mit Unit Tests testen.

Zusammenfassung

- Funktionen haben viele Vorteile.

Zusammenfassung

- Funktionen haben viele Vorteile.
- Wir haben diese nun ziemlich detailiert kennengelernt.

Zusammenfassung

- Funktionen haben viele Vorteile.
- Wir haben diese nun ziemlich detailiert kennengelernt.
- Wir haben Funktionen implementiert und getestet.

Zusammenfassung

- Funktionen haben viele Vorteile.
- Wir haben diese nun ziemlich detailiert kennengelernt.
- Wir haben Funktionen implementiert und getestet.
- Sie sind einer der wichtigsten Bausteine des Kontrollflusses.

Zusammenfassung

- Funktionen haben viele Vorteile.
- Wir haben diese nun ziemlich detailiert kennengelernt.
- Wir haben Funktionen implementiert und getestet.
- Sie sind einer der wichtigsten Bausteine des Kontrollflusses.
- Es gibt aber noch einen weiteren wichtigen Baustein...

谢谢您们！

Thank you!

Vielen Dank!

References I

[1] Milton Abramowitz und Irene A. Stegun, Hrsg. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Tenth Printing, with corrections. Bd. 55 der Reihe National Bureau of Standards Applied Mathematics Series. Gaithersburg, MD, USA: United States Department of Commerce, National Bureau of Standards und Washington, D.C., USA: United States Government Printing Office, Juni 1964–Dez. 1972. ISSN: 0083-1786. ISBN: 978-0-16-000202-1. URL: https://personal.math.ubc.ca/~cbm/aands (besucht am 2025-09-05) (siehe S. 61–138, 174).

[2] "Annotating `Callable` Objects". In: *Python 3 Documentation. The Python Standard Library*. Beaverton, OR, USA: Python Software Foundation (PSF), 2001–2025. URL: https://docs.python.org/3/library/typing.html#annotating-callables (besucht am 2024-10-09) (siehe S. 61–81).

[3] Tom Mike Apostol. "Primitive Functions and the Second Fundamental Theorem of Calculus". In: *One-Variable Calculus, with an Introduction to Linear Algebra*. 2. Aufl. Bd. 1 der Reihe Calculus. Chichester, West Sussex, England, UK: John Wiley and Sons Ltd., Jan. 1991. Kap. 5.3, S. 205–207. ISBN: 978-0-471-00005-1 (siehe S. 19–26).

[4] Adam Aspin und Karine Aspin. *Query Answers with MariaDB – Volume I: Introduction to SQL Queries*. Tetras Publishing, Okt. 2018. ISBN: 978-1-9996172-4-0. See also[5] (siehe S. 160, 173).

[5] Adam Aspin und Karine Aspin. *Query Answers with MariaDB – Volume II: In-Depth Querying*. Tetras Publishing, Okt. 2018. ISBN: 978-1-9996172-5-7. See also[4] (siehe S. 160, 173).

[6] Kendall Eugene Atkinson. *An Introduction to Numerical Analysis*. 2. Aufl. Chichester, West Sussex, England, UK: John Wiley and Sons Ltd., 1991. ISBN: 978-0-471-62489-9 (siehe S. 38–40).

[7] Daniel J. Barrett. *Efficient Linux at the Command Line*. Sebastopol, CA, USA: O'Reilly Media, Inc., Feb. 2022. ISBN: 978-1-0981-1340-7 (siehe S. 173, 175).

[8] Daniel Bartholomew. *Learning the MariaDB Ecosystem: Enterprise-level Features for Scalability and Availability*. New York, NY, USA: Apress Media, LLC, Okt. 2019. ISBN: 978-1-4842-5514-8 (siehe S. 173).

[9] Kent L. Beck. *JUnit Pocket Guide*. Sebastopol, CA, USA: O'Reilly Media, Inc., Sep. 2004. ISBN: 978-0-596-00743-0 (siehe S. 176).

References II

[10] Tim Berners-Lee. *Re: Qualifiers on Hypertext links...* Geneva, Switzerland: World Wide Web project, European Organization for Nuclear Research (CERN) und Newsgroups: alt.hypertext, 6. Aug. 1991. URL: https://www.w3.org/People/Berners-Lee/1991/08/art-6484.txt (besucht am 2025-02-05) (siehe S. 176).

[11] Alex Berson. *Client/Server Architecture*. 2. Aufl. Computer Communications Series. New York, NY, USA: McGraw-Hill, 29. März 1996. ISBN: 978-0-07-005664-0 (siehe S. 172).

[12] Silvia Botros und Jeremy Tinley. *High Performance MySQL*. 4. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., Nov. 2021. ISBN: 978-1-4920-8051-0 (siehe S. 174).

[13] Ed Bott. *Windows 11 Inside Out*. Hoboken, NJ, USA: Microsoft Press, Pearson Education, Inc., Feb. 2023. ISBN: 978-0-13-769132-6 (siehe S. 173).

[14] Ron Brash und Ganesh Naik. *Bash Cookbook*. Birmingham, England, UK: Packt Publishing Ltd, Juli 2018. ISBN: 978-1-78862-936-2 (siehe S. 172).

[15] Florian Bruhin. *Python f-Strings*. Winterthur, Switzerland: Bruhin Software, 31. Mai 2023. URL: https://fstring.help (besucht am 2024-07-25) (siehe S. 173).

[16] Jason Cannon. *High Availability for the LAMP Stack*. Shelter Island, NY, USA: Manning Publications, Juni 2022 (siehe S. 173, 174).

[17] Donald D. Chamberlin. "50 Years of Queries". *Communications of the ACM (CACM)* 67(8):110–121, Aug. 2024. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0001-0782. doi:10.1145/3649887. URL: https://cacm.acm.org/research/50-years-of-queries (besucht am 2025-01-09) (siehe S. 174).

[18] David Clinton und Christopher Negus. *Ubuntu Linux Bible*. 10. Aufl. Bible Series. Chichester, West Sussex, England, UK: John Wiley and Sons Ltd., 10. Nov. 2020. ISBN: 978-1-119-72233-5 (siehe S. 175).

[19] Edgar Frank „Ted" Codd. "A Relational Model of Data for Large Shared Data Banks". *Communications of the ACM (CACM)* 13(6):377–387, Juni 1970. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0001-0782. doi:10.1145/362384.362685. URL: https://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf (besucht am 2025-01-05) (siehe S. 174).

References III

[20] "Data Model". In: *Python 3 Documentation. The Python Language Reference*. Beaverton, OR, USA: Python Software Foundation (PSF), 2001–2025. Kap. 3. URL: https://docs.python.org/3/reference/datamodel.html (besucht am 2024-08-22) (siehe S. 5–17).

[21] *Database Language SQL*. Techn. Ber. ANSI X3.135-1986. Washington, D.C., USA: American National Standards Institute (ANSI), 1986 (siehe S. 174).

[22] Matt David und Blake Barnhill. *How to Teach People SQL*. San Francisco, CA, USA: The Data School, Chart.io, Inc., 10. Dez. 2019–10. Apr. 2023. URL: https://dataschool.com/how-to-teach-people-sql (besucht am 2025-02-27) (siehe S. 174).

[23] *Database Language SQL*. International Standard ISO 9075-1987. Geneva, Switzerland: International Organization for Standardization (ISO), 1987 (siehe S. 174).

[24] Paul Deitel, Harvey Deitel und Abbey Deitel. *Internet & World Wide WebW[: How to Program*. 5. Aufl. Hoboken, NJ, USA: Pearson Education, Inc., Nov. 2011. ISBN: 978-0-13-299045-5 (siehe S. 176).

[25] Slobodan Dmitrović. *Modern C for Absolute Beginners: A Friendly Introduction to the C Programming Language*. New York, NY, USA: Apress Media, LLC, März 2024. ISBN: 979-8-8688-0224-9 (siehe S. 172).

[26] Russell J.T. Dyer. *Learning MySQL and MariaDB*. Sebastopol, CA, USA: O'Reilly Media, Inc., März 2015. ISBN: 978-1-4493-6290-4 (siehe S. 173, 174).

[27] ."infinitesimal". In: *Merriam-Webster: America's Most Trusted Dictionary*. Hrsg. von Editors of Merriam-Webster. 4. Sep. 2025. URL: https://www.merriam-webster.com/dictionary/infinitesimal (besucht am 2025-09-08) (siehe S. 19–26, 173).

[28] James F. Epperson. *An Introduction to Numerical Methods and Analysis*. 2. Aufl. John Wiley and Sons Ltd., Okt. 2013. ISBN: 978-1-118-36759-9 (siehe S. 38–40, 61–138).

[29] "Escape Sequences". In: *Python 3 Documentation. The Python Language Reference*. Beaverton, OR, USA: Python Software Foundation (PSF), 2001–2025. Kap. 2.4.1.1. URL: https://docs.python.org/3/reference/lexical_analysis.html#escape-sequences (besucht am 2025-08-05) (siehe S. 172).

References IV

[30] Leonhard Euler. "An Essay on Continued Fractions". Übers. von Myra F. Wyman und Bostwick F. Wyman. *Mathematical Systems Theory* 18(1):295–328, Dez. 1985. New York, NY, USA: Springer Science+Business Media, LLC. ISSN: 1432-4350. doi:10.1007/BF01699475. URL: https://www.researchgate.net/publication/301720080 (besucht am 2024-09-24). Translation of[31]. (Siehe S. 163).

[31] Leonhard Euler. "De Fractionibus Continuis Dissertation". *Commentarii Academiae Scientiarum Petropolitanae* 9:98–137, 1737–1744. Petropolis (St. Petersburg), Russia: Typis Academiae. URL: https://scholarlycommons.pacific.edu/cgi/viewcontent.cgi?article=1070 (besucht am 2024-09-24). See[30] for a translation. (Siehe S. 163, 176).

[32] Merran Evans, Nicholas Hastings und Brian Peacock. *Statistical Distributions*. 3. Aufl. Chichester, West Sussex, England, UK: Wiley Interscience, Juni 2000. ISBN: 978-0-471-37124-3 (siehe S. 61–138, 174).

[33] Luca Ferrari und Enrico Pirozzi. *Learn PostgreSQL*. 2. Aufl. Birmingham, England, UK: Packt Publishing Ltd, Okt. 2023. ISBN: 978-1-83763-564-1 (siehe S. 174).

[34] Michael Filaseta. "The Transcendence of e and π". In: *Math 785: Transcendental Number Theory*. Columbia, SC, USA: University of South Carolina, Frühling 2011. Kap. 6. URL: https://people.math.sc.edu/filaseta/gradcourses/Math785/Math785Notes6.pdf (besucht am 2024-07-05) (siehe S. 176).

[35] "Formatted String Literals". In: *Python 3 Documentation. The Python Tutorial*. Beaverton, OR, USA: Python Software Foundation (PSF), 2001–2025. Kap. 7.1.1. URL: https://docs.python.org/3/tutorial/inputoutput.html#formatted-string-literals (besucht am 2024-07-25) (siehe S. 173).

[36] Bhavesh Gawade. "Mastering F-Strings in Python: Efficient String Handling in Python Using Smart F-Strings". In: *C O D E B*. Mumbai, Maharashtra, India: Code B Solutions Pvt Ltd, 25. Apr.–3. Juni 2025. URL: https://code-b.dev/blog/f-strings-in-python (besucht am 2025-08-04) (siehe S. 173).

[37] David Goodger und Guido van Rossum. *Docstring Conventions*. Python Enhancement Proposal (PEP) 257. Beaverton, OR, USA: Python Software Foundation (PSF), 29. Mai–13. Juni 2001. URL: https://peps.python.org/pep-0257 (besucht am 2024-07-27) (siehe S. 172).

References V

[38] Olaf Górski. "Why f-strings are awesome: Performance of different string concatenation methods in Python". In: *DEV Community*. Sacramento, CA, USA: DEV Community Inc., 8. Nov. 2022. URL: https://dev.to/grski/performance-of-different-string-concatenation-methods-in-python-why-f-strings-are-awesome-2e97 (besucht am 2025-08-04) (siehe S. 173).

[39] Terry Halpin und Tony Morgan. *Information Modeling and Relational Databases*. 3. Aufl. Burlington, MA, USA/San Mateo, CA, USA: Morgan Kaufmann Publishers, Juli 2024. ISBN: 978-0-443-23791-1 (siehe S. 174).

[40] Jan L. Harrington. *Relational Database Design and Implementation*. 4. Aufl. Burlington, MA, USA/San Mateo, CA, USA: Morgan Kaufmann Publishers, Apr. 2016. ISBN: 978-0-12-849902-3 (siehe S. 174).

[41] Michael Hausenblas. *Learning Modern Linux*. Sebastopol, CA, USA: O'Reilly Media, Inc., Apr. 2022. ISBN: 978-1-0981-0894-6 (siehe S. 173).

[42] Matthew Helmke. *Ubuntu Linux Unleashed 2021 Edition*. 14. Aufl. Reading, MA, USA: Addison-Wesley Professional, Aug. 2020. ISBN: 978-0-13-668539-5 (siehe S. 173, 175).

[43] Trey Hunner. "Everything is an Object". In: *Python Morsels*. Reykjavík, Iceland: Python Morsels, 25. Feb. 2021. URL: https://www.pythonmorsels.com/everything-is-an-object (besucht am 2025-09-08) (siehe S. 5–17).

[44] John Hunt. *A Beginners Guide to Python 3 Programming*. 2. Aufl. Undergraduate Topics in Computer Science (UTICS). Cham, Switzerland: Springer, 2023. ISBN: 978-3-031-35121-1. doi:10.1007/978-3-031-35122-8 (siehe S. 174).

[45] *Information Technology – Database Languages – SQL – Part 1: Framework (SQL/Framework), Part 1*. International Standard ISO/IEC 9075-1:2023(E), Sixth Edition, (ANSI X3.135). Geneva, Switzerland: International Organization for Standardization (ISO) und International Electrotechnical Commission (IEC), Juni 2023. URL: https://standards.iso.org/ittf/PubliclyAvailableStandards/ISO_IEC_9075-1_2023_ed_6_-_id_76583_Publication_PDF_(en).zip (besucht am 2025-01-08). Consists of several parts, see https://modern-sql.com/standard for information where to obtain them. (Siehe S. 174).

[46] *Information Technology – Universal Coded Character Set (UCS)*. International Standard ISO/IEC 10646:2020. Geneva, Switzerland: International Organization for Standardization (ISO) und International Electrotechnical Commission (IEC), Dez. 2020 (siehe S. 175).

References VI

[47] Steven G. Johnson. "Numerical Integration and the Redemption of the Trapezoidal Rule". In: *18.335 Introduction to Numerical Methods*. Cambridge, MA, USA: Massachusetts Institute of Technology (MIT), Frühling 2011–8. Mai 2020. URL: https://math.mit.edu/%7Estevenj/trap-iap-2011.pdf (besucht am 2025-09-07). Past lectures are available at https://github.com/mitmath/18335 (visited on 2025-09-07) (siehe S. 38–40).

[48] Arthur Jones, Kenneth R. Pearson und Sidney A. Morris. "Transcendence of e and π". In: *Abstract Algebra and Famous Impossibilities*. Universitext (UTX). New York, NY, USA: Springer New York, 1991. Kap. 9, S. 115–161. ISSN: 0172-5939. ISBN: 978-1-4419-8552-1. doi:10.1007/978-1-4419-8552-1_8 (siehe S. 176).

[49] Mikhail G. Katz und David Sherry. "Leibniz's Infinitesimals: Their Fictionality, Their Modern Implementations, and Their Foes from Berkeley to Russell and Beyond". *Erkenntnis: An International Journal of Scientific Philosophy* 78(3):571–625, Juni 2013. London, England, UK: Springer Nature Limited. ISSN: 0165-0106. doi:10.1007/s10670-012-9370-y (siehe S. 19–26, 173).

[50] Jay LaCroix. *Mastering Ubuntu Server*. 4. Aufl. Birmingham, England, UK: Packt Publishing Ltd, Sep. 2022. ISBN: 978-1-80323-424-3 (siehe S. 174).

[51] "Lambda". In: *Python 3 Documentation. The Python Language Reference*. Beaverton, OR, USA: Python Software Foundation (PSF), 2001–2025. Kap. 6.14. URL: https://docs.python.org/3/reference/expressions.html#lambda (besucht am 2024-10-09) (siehe S. 61–97).

[52] Łukasz Langa. *Literature Overview for Type Hints*. Python Enhancement Proposal (PEP) 482. Beaverton, OR, USA: Python Software Foundation (PSF), 8. Jan. 2015. URL: https://peps.python.org/pep-0482 (besucht am 2024-10-09) (siehe S. 175).

[53] Kent D. Lee und Steve Hubbard. *Data Structures and Algorithms with Python*. Undergraduate Topics in Computer Science (UTICS). Cham, Switzerland: Springer, 2015. ISBN: 978-3-319-13071-2. doi:10.1007/978-3-319-13072-9 (siehe S. 174).

[54] Jukka Lehtosalo, Ivan Levkivskyi, Jared Hance, Ethan Smith, Guido van Rossum, Jelle „JelleZijlstra" Zijlstra, Michael J. Sullivan, Shantanu Jain, Xuanda Yang, Jingchen Ye, Nikita Sobolev und Mypy Contributors. *Mypy – Static Typing for Python*. San Francisco, CA, USA: GitHub Inc, 2024. URL: https://github.com/python/mypy (besucht am 2024-08-17) (siehe S. 173).

References VII

[55] Gloria Lotha, Aakanksha Gaur, Erik Gregersen, Swati Chopra und William L. Hosch. "Client-Server Architecture". In: *Encyclopaedia Britannica*. Hrsg. von The Editors of Encyclopaedia Britannica. Chicago, IL, USA: Encyclopædia Britannica, Inc., 3. Jan. 2025. URL: https://www.britannica.com/technology/client-server-architecture (besucht am 2025-01-20) (siehe S. 172).

[56] Mark Lutz. *Learning Python*. 6. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., März 2025. ISBN: 978-1-0981-7130-8 (siehe S. 174).

[57] *MariaDB Server Documentation*. Milpitas, CA, USA: MariaDB, 2025. URL: https://mariadb.com/kb/en/documentation (besucht am 2025-04-24) (siehe S. 173).

[58] "Mathematical Functions and Operators". In: *PostgreSQL Documentation*. 17.4. The PostgreSQL Global Development Group (PGDG), 20. Feb. 2025. Kap. 9.3. URL: https://www.postgresql.org/docs/17/functions-math.html (besucht am 2025-02-27) (siehe S. 176).

[59] Aaron Maxwell. *What are f-strings in Python and how can I use them?* Oakville, ON, Canada: Infinite Skills Inc, Juni 2017. ISBN: 978-1-4919-9486-3 (siehe S. 173).

[60] Jim Melton und Alan R. Simon. *SQL: 1999 – Understanding Relational Language Components*. The Morgan Kaufmann Series in Data Management Systems. Burlington, MA, USA/San Mateo, CA, USA: Morgan Kaufmann Publishers, Juni 2001. ISBN: 978-1-55860-456-8 (siehe S. 174).

[61] Mark Mendoza. *Parameter Specification Variables*. Python Enhancement Proposal (PEP) 612. Beaverton, OR, USA: Python Software Foundation (PSF), 18. Dez. 2019–13. Juli 2020. URL: https://peps.python.org/pep-0612 (besucht am 2024-10-09) (siehe S. 61–75).

[62] Cameron Newham und Bill Rosenblatt. *Learning the Bash Shell – Unix Shell Programming: Covers Bash 3.0*. 3. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., 2005. ISBN: 978-0-596-00965-6 (siehe S. 172).

[63] Ivan Niven. "The Transcendence of π". *The American Mathematical Monthly* 46(8):469–471, Okt. 1939. London, England, UK: Taylor and Francis Ltd. ISSN: 1930-0972. doi:10.2307/2302515 (siehe S. 176).

[64] Regina O. Obe und Leo S. Hsu. *PostgreSQL: Up and Running*. 3. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., Okt. 2017. ISBN: 978-1-4919-6336-4 (siehe S. 174).

References VIII

[65] A. Jefferson Offutt. "Unit Testing Versus Integration Testing". In: *Test: Faster, Better, Sooner – IEEE International Test Conference (ITC'1991)*. 26.–30. Okt. 1991, Nashville, TN, USA. Los Alamitos, CA, USA: IEEE Computer Society, 1991. Kap. Paper P2.3, S. 1108–1109. ISSN: 1089-3539. ISBN: 978-0-8186-9156-0. doi:10.1109/TEST.1991.519784 (siehe S. 176).

[66] Michael Olan. "Unit Testing: Test Early, Test Often". *Journal of Computing Sciences in Colleges (JCSC)* 19(2):319–328, Dez. 2003. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 1937-4771. doi:10.5555/948785.948830. URL: https://www.researchgate.net/publication/255673967 (besucht am 2025-09-05) (siehe S. 176).

[67] Robert Orfali, Dan Harkey und Jeri Edwards. *Client/Server Survival Guide*. 3. Aufl. Chichester, West Sussex, England, UK: John Wiley and Sons Ltd., 25. Jan. 1999. ISBN: 978-0-471-31615-2 (siehe S. 172).

[68] Ashwin Pajankar. *Python Unit Test Automation: Automate, Organize, and Execute Unit Tests in Python*. New York, NY, USA: Apress Media, LLC, Dez. 2021. ISBN: 978-1-4842-7854-3 (siehe S. 176).

[69] Yasset Pérez-Riverol, Laurent Gatto, Rui Wang, Timo Sachsenberg, Julian Uszkoreit, Felipe da Veiga Leprevost, Christian Fufezan, Tobias Ternent, Stephen J. Eglen, Daniel S. Katz, Tom J. Pollard, Alexander Konovalov, Robert M. Flight, Kai Blin und Juan Antonio Vizcaíno. "Ten Simple Rules for Taking Advantage of Git and GitHub". *PLOS Computational Biology* 12(7), 14. Juli 2016. San Francisco, CA, USA: Public Library of Science (PLOS). ISSN: 1553-7358. doi:10.1371/JOURNAL.PCBI.1004947 (siehe S. 173).

[70] *PostgreSQL Documentation*. 17.4. The PostgreSQL Global Development Group (PGDG), Feb. 2025. URL: https://www.postgresql.org/docs/17/index.html (besucht am 2025-02-25).

[71] *PostgreSQL Essentials: Leveling Up Your Data Work*. Sebastopol, CA, USA: O'Reilly Media, Inc., März 2024 (siehe S. 174).

[72] Philippe PRADOS und Maggie Moss. *Allow Writing Union Types as `X | Y`*. Python Enhancement Proposal (PEP) 604. Beaverton, OR, USA: Python Software Foundation (PSF), 28. Aug. 2019. URL: https://peps.python.org/pep-0604 (besucht am 2024-10-09) (siehe S. 61–79).

[73] *Programming Languages – C, Working Document of SC22/WG14*. International Standard ISO/3IEC9899:2017 C17 Ballot N2176. Geneva, Switzerland: International Organization for Standardization (ISO) und International Electrotechnical Commission (IEC), Nov. 2017. URL: https://files.lhmouse.com/standards/ISO%20C%20N2176.pdf (besucht am 2024-06-29) (siehe S. 172).

References IX

[74] Abhishek Ratan, Eric Chou, Pradeeban Kathiravelu und Dr. M.O. Faruque Sarker. *Python Network Programming*. Birmingham, England, UK: Packt Publishing Ltd, Jan. 2019. ISBN: 978-1-78883-546-6 (siehe S. 172).

[75] Federico Razzoli. *Mastering MariaDB*. Birmingham, England, UK: Packt Publishing Ltd, Sep. 2014. ISBN: 978-1-78398-154-0 (siehe S. 173).

[76] Mike Reichardt, Michael Gundall und Hans D. Schotten. "Benchmarking the Operation Times of NoSQL and MySQL Databases for Python Clients". In: *47th Annual Conference of the IEEE Industrial Electronics Society (IECON'2021*. 13.–15. Okt. 2021, Toronto, ON, Canada. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE), 2021, S. 1–8. ISSN: 2577-1647. ISBN: 978-1-6654-3554-3. doi:10.1109/IECON48115.2021.9589382 (siehe S. 174).

[77] Mark Richards und Neal Ford. *Fundamentals of Software Architecture: An Engineering Approach*. Sebastopol, CA, USA: O'Reilly Media, Inc., Jan. 2020. ISBN: 978-1-4920-4345-4 (siehe S. 172).

[78] Per Runeson. "A Survey of Unit Testing Practices". *IEEE Software* 23(4):22–29, Juli–Aug. 2006. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE). ISSN: 0740-7459. doi:10.1109/MS.2006.91 (siehe S. 176).

[79] Ellen Siever, Stephen Figgins, Robert Love und Arnold Robbins. *Linux in a Nutshell*. 6. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., Sep. 2009. ISBN: 978-0-596-15448-6 (siehe S. 173).

[80] Anna Skoulikari. *Learning Git*. Sebastopol, CA, USA: O'Reilly Media, Inc., Mai 2023. ISBN: 978-1-0981-3391-7 (siehe S. 173).

[81] Eric V. „ericvsmith" Smith. *Literal String Interpolation*. Python Enhancement Proposal (PEP) 498. Beaverton, OR, USA: Python Software Foundation (PSF), 6. Nov. 2016–9. Sep. 2023. URL: https://peps.python.org/pep-0498 (besucht am 2024-07-25) (siehe S. 173).

[82] John Miles Smith und Philip Yen-Tang Chang. "Optimizing the Performance of a Relational Algebra Database Interface". *Communications of the ACM (CACM)* 18(10):568–579, Okt. 1975. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0001-0782. doi:10.1145/361020.361025 (siehe S. 174).

[83] "SQL Commands". In: *PostgreSQL Documentation*. 17.4. The PostgreSQL Global Development Group (PGDG), 20. Feb. 2025. Kap. Part VI. Reference. URL: https://www.postgresql.org/docs/17/sql-commands.html (besucht am 2025-02-25) (siehe S. 174).

References X

[84] Ryan K. Stephens und Ronald R. Plew. *Sams Teach Yourself SQL in 21 Days*. 4. Aufl. Sams Tech Yourself. Indianapolis, IN, USA: SAMS Technical Publishing und Hoboken, NJ, USA: Pearson Education, Inc., Okt. 2002. ISBN: 978-0-672-32451-2 (siehe S. 169, 174).

[85] Ryan K. Stephens, Ronald R. Plew, Bryan Morgan und Jeff Perkins. *SQL in 21 Tagen. Die Datenbank-Abfragesprache SQL vollständig erklärt (in 14/21 Tagen)*. 6. Aufl. Burgthann, Bayern, Germany: Markt+Technik Verlag GmbH, Feb. 1998. ISBN: 978-3-8272-2020-2. Translation of[84] (siehe S. 174).

[86] "String Constants". In: Kap. 4.1.2.1. URL: https://www.postgresql.org/docs/17/sql-syntax-lexical.html#SQL-SYNTAX-STRINGS (besucht am 2025-08-23) (siehe S. 172).

[87] Allen Taylor. *Introducing SQL and Relational Databases*. New York, NY, USA: Apress Media, LLC, Sep. 2018. ISBN: 978-1-4842-3841-7 (siehe S. 174).

[88] Alkin Tezuysal und Ibrar Ahmed. *Database Design and Modeling with PostgreSQL and MySQL*. Birmingham, England, UK: Packt Publishing Ltd, Juli 2024. ISBN: 978-1-80323-347-5 (siehe S. 174).

[89] *Python 3 Documentation. The Python Language Reference*. Beaverton, OR, USA: Python Software Foundation (PSF), 2001–2025. URL: https://docs.python.org/3/reference (besucht am 2025-04-27).

[90] *The Unicode Standard, Version 15.1: Archived Code Charts*. South San Francisco, CA, USA: The Unicode Consortium, 25. Aug. 2023. URL: https://www.unicode.org/Public/15.1.0/charts/CodeCharts.pdf (besucht am 2024-07-26) (siehe S. 175).

[91] George K. Thiruvathukal, Konstantin Läufer und Benjamin Gonzalez. "Unit Testing Considered Useful". *Computing in Science & Engineering* 8(6):76–87, Nov.–Dez. 2006. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE). ISSN: 1521-9615. doi:10.1109/MCSE.2006.124. URL: https://www.researchgate.net/publication/220094077 (besucht am 2024-10-01) (siehe S. 176).

[92] Linus Torvalds. "The Linux Edge". *Communications of the ACM (CACM)* 42(4):38–39, Apr. 1999. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0001-0782. doi:10.1145/299157.299165 (siehe S. 173).

References XI

[93] Mariot Tsitoara. *Beginning Git and GitHub: Version Control, Project Management and Teamwork for the New Developer*. New York, NY, USA: Apress Media, LLC, März 2024. ISBN: 979-8-8688-0215-7 (siehe S. 173, 176).

[94] *Unicode®15.1.0*. South San Francisco, CA, USA: The Unicode Consortium, 12. Sep. 2023. ISBN: 978-1-936213-33-7. URL: https://www.unicode.org/versions/Unicode15.1.0 (besucht am 2024-07-26) (siehe S. 175).

[95] Guido van Rossum und Łukasz Langa. *Type Hints*. Python Enhancement Proposal (PEP) 484. Beaverton, OR, USA: Python Software Foundation (PSF), 29. Sep. 2014. URL: https://peps.python.org/pep-0484 (besucht am 2024-08-22) (siehe S. 175).

[96] Guido van Rossum, Barry Warsaw und Alyssa Coghlan. *Style Guide for Python Code*. Python Enhancement Proposal (PEP) 8. Beaverton, OR, USA: Python Software Foundation (PSF), 5. Juli 2001. URL: https://peps.python.org/pep-0008 (besucht am 2024-07-27) (siehe S. 172).

[97] Sander van Vugt. *Linux Fundamentals*. 2. Aufl. Hoboken, NJ, USA: Pearson IT Certification, Juni 2022. ISBN: 978-0-13-792931-3 (siehe S. 173).

[98] Christian Walck. *Hand-Book on Statistical Distributions for Experimentalists*. Internal Report SUF-PFY/96-01. Stockholm, Sweden: University of Stockholm, 11. Dez. 1996–10. Sep. 2007. URL: https://www.stat.rice.edu/~dobelman/textfiles/DistributionsHandbook.pdf (besucht am 2025-09-05) (siehe S. 61–138, 174).

[99] Thomas Weise (汤卫思). *Databases*. Hefei, Anhui, China (中国安徽省合肥市): Hefei University (合肥大学), School of Artificial Intelligence and Big Data (人工智能与大数据学院), Institute of Applied Optimization (应用优化研究所, IAO), 2025. URL: https://thomasweise.github.io/databases (besucht am 2025-01-05) (siehe S. 172, 174).

[100] Thomas Weise (汤卫思). *Programming with Python*. Hefei, Anhui, China (中国安徽省合肥市): Hefei University (合肥大学), School of Artificial Intelligence and Big Data (人工智能与大数据学院), Institute of Applied Optimization (应用优化研究所, IAO), 2024–2025. URL: https://thomasweise.github.io/programmingWithPython (besucht am 2025-01-05) (siehe S. 172–174).

[101] Eric Wolfgang Weisstein. "Second Fundamental Theorem of Calculus". In: *MathWorld – A Wolfram Web Resource*. Champaign, IL, USA: Wolfram Research, Inc., 4. Sep. 2025. URL: https://mathworld.wolfram.com/SecondFundamentalTheoremofCalculus.html (besucht am 2025-09-08) (siehe S. 19–26).

References XII

[102] *What is a Relational Database?* Armonk, NY, USA: International Business Machines Corporation (IBM), 20. Okt. 2021–12. Dez. 2024. URL: https://www.ibm.com/think/topics/relational-databases (besucht am 2025-01-05) (siehe S. 174).

[103] Ulf Michael „Monty“ Widenius, David Axmark und Uppsala, Sweden: MySQL AB. *MySQL Reference Manual – Documentation from the Source*. Sebastopol, CA, USA: O'Reilly Media, Inc., 9. Juli 2002. ISBN: 978-0-596-00265-7 (siehe S. 174).

[104] Kinza Yasar und Craig S. Mullins. *Definition: Database Management System (DBMS)*. Newton, MA, USA: TechTarget, Inc., Juni 2024. URL: https://www.techtarget.com/searchdatamanagement/definition/database-management-system (besucht am 2025-01-11) (siehe S. 172).

[105] Giorgio Zarrelli. *Mastering Bash*. Birmingham, England, UK: Packt Publishing Ltd, Juni 2017. ISBN: 978-1-78439-687-9 (siehe S. 172).

Glossary (in English) I

Bash is a the shell used under Ubuntu Linux, i.e., the program that „runs“ in the terminal and interprets your commands, allowing you to start and interact with other programs[14,62,105]. Learn more at https://www.gnu.org/software/bash.

C is a programming language, which is very successful in system programming situations[25,73].

client In a client-server architecture, the client is a device or process that requests a service from the server. It initiates the communication with the server, sends a request, and receives the response with the result of the request. Typical examples for clients are web browsers in the internet as well as clients for database management systems (DBMSes), such as psql.

client-server architecture is a system design where a central server receives requests from one or multiple clients[11,55,67,74,77]. These requests and responses are usually sent over network connections. A typical example for such a system is the World Wide Web (WWW), where web servers host websites and make them available to web browsers, the clients. Another typical example is the structure of database (DB) software, where a central server, the DBMS, offers access to the DB to the different clients. Here, the client can be some terminal software shipping with the DBMS, such as psql, or the different applications that access the DBs.

DB A *database* is an organized collection of structured information or data, typically stored electronically in a computer system. Databases are discussed in our book *Databases*[99].

DBMS A *database management system* is the software layer located between the user or application and the DB. The DBMS allows the user/application to create, read, write, update, delete, and otherwise manipulate the data in the DB[104].

docstring Docstrings are special string constants in Python that contain documentation for modules or functions[37]. They must be delimited by `"""..."""`[37,96].

escape sequence Escaping is the process of presenting „forbidden“ characters or symbols in a sequence of characters or symbols. In Python[100], string escapes allow us to include otherwise impossible characters, such as string delimiters, in a string. Each such character is represented by an *escape sequence*, which usually starts with the backslash character („\“)[29]. In Python strings, the escape sequence `\"`, for example, stands for `"`, the escape sequence `\\` stands for `\`, and the escape sequence `\n` stands for a newline or linebreak character. In Python f-strings, the escape sequence `{{` stands for a single curly brace `{`. In PostgreSQL[99], similar C-style escapes (starting with „\“) are supported[86].

Glossary (in English) II

f-string let you include the results of expressions in strings[15,35,36,38,59,81]. They can contain expressions (in curly braces) like `f"a{6-1}b"` that are then transformed to text via (string) interpolation, which turns the string to `"a5b"`. F-strings are delimited by `f"..."`.

Git is a distributed Version Control Systems (VCS) which allows multiple users to work on the same code while preserving the history of the code changes[80,93]. Learn more at https://git-scm.com.

GitHub is a website where software projects can be hosted and managed via the Git VCS[69,93]. Learn more at https://github.com.

infinitesimal an infinitesimal value is immeasurably or incalculably small, i.e., arbitrarily close to but greater than zero[27,49].

IT information technology

LAMP Stack A system setup for web applications: Linux, Apache (a web server), MySQL, and the server-side scripting language PHP[16,42].

Linux is the leading open source operating system, i.e., a free alternative for Microsoft Windows[7,41,79,92,97]. We recommend using it for this course, for software development, and for research. Learn more at https://www.linux.org. Its variant Ubuntu is particularly easy to use and install.

MariaDB An open source relational database management system that has forked off from MySQL[4,5,8,26,57,75]. See https://mariadb.org for more information.

Microsoft Windows is a commercial proprietary operating system[13]. It is widely spread, but we recommend using a Linux variant such as Ubuntu for software development and for our course. Learn more at https://www.microsoft.com/windows.

Mypy is a static type checking tool for Python[54] that makes use of type hints. Learn more at https://github.com/python/mypy and in[100].

Glossary (in English) III

MySQL An open source relational database management system[12,26,76,88,103]. MySQL is famous for its use in the LAMP Stack. See https://www.mysql.com for more information.

PDF *Probability Density Function*[1,32,98] of a continuous random variable $\mathcal{X}$ that can take on values from range $\Omega \subseteq \mathbb{R}$ is a function $f_{\mathcal{X}} : \Omega \mapsto \mathbb{R}^+$ such that

- $f_{\mathcal{X}}(x) > 0$ for all $x \in \Omega$ (non-negativity),
- $\int_{\Omega} f_{\mathcal{X}}(x)dx = 1$ (normalization), and
- the probability that $x \in A$ for all $A \subseteq \Omega$ is $\int_A f_{\mathcal{X}}(x)dx$.

PostgreSQL An open source object-relational DBMS[33,64,71,88]. See https://postgresql.org for more information.

psql is the client program used to access the PostgreSQL DBMS server.

Python The Python programming language[44,53,56,100], i.e., what you will learn about in our book[100]. Learn more at https://python.org.

relational database A relational DB is a database that organizes data into rows (tuples, records) and columns (attributes), which collectively form tables (relations) where the data points are related to each other[19,39,40,82,87,99,102].

server In a client-server architecture, the server is a process that fulfills the requests of the clients. It usually waits for incoming communication carring the requests from the clients. For each request, it takes the necessary actions, performs the required computations, and then sends a response with the result of the request. Typical examples for servers are web servers[16] in the internet as well as DBMSes. It is also common to refer to the computer running the server processes as server as well, i.e., to call it the „server computer“[50].

SQL The *Structured Query Language* is basically a programming language for querying and manipulating relational databases[17,21–23,45,60,83–85,87]. It is understood by many DBMSes. You find the Structured Query Language (SQL) commands supported by PostgreSQL in the reference[83].

Glossary (in English) IV

(string) interpolation In Python, string interpolation is the process where all the expressions in an f-string are evaluated and the final string is constructed. An example for string interpolation is turning `f"Rounded {1.234:.2f}"` to `"Rounded 1.23"`.

terminal A terminal is a text-based window where you can enter commands and execute them[7,18]. Knowing what a terminal is and how to use it is very essential in any programming- or system administration-related task. If you want to open a terminal under Microsoft Windows, you can Druck auf ⊞+R, dann Schreiben von `cmd`, dann Druck auf ↵. Under Ubuntu Linux, Ctrl+Alt+T opens a terminal, which then runs a Bash shell inside.

type hint are annotations that help programmers and static code analysis tools such as Mypy to better understand what type a variable or function parameter is supposed to be[52,95]. Python is a dynamically typed programming language where you do not need to specify the type of, e.g., a variable. This creates problems for code analysis, both automated as well as manual: For example, it may not always be clear whether a variable or function parameter should be an integer or floating point number. The annotations allow us to explicitly state which type is expected. They are *ignored* during the program execution. They are a basically a piece of documentation.

Ubuntu is a variant of the open source operating system Linux[18,42]. We recommend that you use this operating system to follow this class, for software development, and for research. Learn more at https://ubuntu.com. If you are in China, you can download it from https://mirrors.ustc.edu.cn/ubuntu-releases.

Unicode A standard for assigning characters to numbers[46,90,94]. The Unicode standard supports basically all characters from all languages that are currently in use, as well as many special symbols. It is the predominantly used way to represent characters in computers and is regularly updated and improved.

Glossary (in English) V

unit test Software development is centered around creating the program code of an application, library, or otherwise useful system. A *unit test* is an *additional* code fragment that is not part of that productive code. It exists to execute (a part of) the productive code in a certain scenario (e.g., with specific parameters), to observe the behavior of that code, and to compare whether this behavior meets the specification[9,65,66,68,78,91]. If not, the unit test fails. The use of unit tests is at least threefold: First, they help us to detect errors in the code. Second, program code is usually not developed only once and, from then on, used without change indefinitely. Instead, programs are often updated, improved, extended, and maintained over a long time. Unit tests can help us to detect whether such changes in the program code, maybe after years, violate the specification or, maybe, cause another, depending, module of the program to violate its specification. Third, they are part of the documentation or even specification of a program.

VCS A *Version Control System* is a software which allows you to manage and preserve the historical development of your program code[93]. A distributed VCS allows multiple users to work on the same code and upload their changes to the server, which then preserves the change history. The most popular distributed VCS is Git.

WWW World Wide Web[10,24]

x-axis The x-axis is the horizontal axis of a two-dimensional coordinate system, often referred to abscissa.

π is the ratio of the circumference U of a circle and its diameter d, i.e., $\pi = U/d$. $\pi \in \mathbb{R}$ is an irrational and transcendental number[34,48,63], which is approximately $\pi \approx 3.141\,592\,653\,589\,793\,238\,462\,643$. In Python, it is provided by the `math` module as constant `pi` with value `3.141592653589793`. In PostgreSQL, it is provided by the SQL function `pi()` with value `3.141592653589793`[58].

e is Euler's number[31], the base of the natural logarithm. $e \in \mathbb{R}$ is an irrational and transcendental number[34,48], which is approximately $e \approx 2.718\,281\,828\,459\,045\,235\,360$. In Python, it is provided by the `math` module as constant `e` with value `2.718281828459045`. In PostgreSQL, you can obtain it via the SQL function `exp(1)` as value `2.718281828459045`[58].

Glossary (in English) VI

$\mathbb{R}$ the set of the real numbers.

$\mathbb{R}^+$ the set of the positive real numbers, i.e., $\mathbb{R}^+ = \{x \in \mathbb{R} : x > 0\}$.