Grundlegende Datentypen

by Woche 2

Zahlen sind nur einer von vielen grundlegenden Datentypen, auf die man in Python stößt, insbesondere wenn man Datenanalysen durchführt. Ein fundiertes Verständnis dieser grundlegenden Datentypen ist entscheidend, um mit Daten in Python zu arbeiten.

Integer (Ganze Zahlen)

Integers (kurz “ints”; Ganze Zahlen), sind numerische Werte ohne Dezimalstellen (siehe Wiki). Jede positive oder negative Zahl (oder 0) ohne Dezimalstelle ist ein Integer in Python. Integers haben eine unbegrenzte Genauigkeit, was bedeutet, dass sie exakt sind. Mit der Funktion `type()` kann der Typ eines Python-Objekts überprüft werden:

```
type(12)
```

```
int
```

Wie man sieht, ist der Typ von 12 “int”. Man kann auch die Funktion `isinstance()` verwenden, um zu überprüfen, ob ein Objekt eine Instanz eines bestimmten Typs ist:

```
isinstance(12, int) # Überprüfen, ob 12 vom Typ "int" ist
```

```
True
```

Der Code bestätigt, dass 12 ein Integer (`int`) ist.

Integer unterstützen alle grundlegenden mathematischen Operationen aus dem letzten Kapitel. Wenn eine mathematische Operation mit Integer zu einem Ergebnis führt, das kein Integer ist, sondern ein Dezimalwert, wird das Ergebnis zu einem Float:

```
1/3
```

```
0.3333333333333333
```

```
type(1/3)
```

```
float
```

Floats (Gleitkommazahlen)

Floats (floating point number; Gleitkommazahlen) sind Zahlen mit Dezimalstellen (siehe Wiki). Im Gegensatz zu Integer haben Floats keine unbegrenzte Genauigkeit, da irrationale Dezimalzahlen unendlich lang sind und daher nicht im Speicher abgelegt werden können. Stattdessen wird der Wert langer Dezimalstellen vom Computer angenähert, sodass es bei langen Floats zu kleinen Rundungsfehlern kommen kann. Dieser Fehler ist so gering, dass er normalerweise nicht von Bedeutung ist, kann sich jedoch in bestimmten Fällen bei vielen wiederholten Berechnungen summieren.

Jede Zahl in Python mit einem Dezimalpunkt ist ein Float, auch wenn nach dem Dezimalpunkt keine von null verschiedenen Zahlen stehen:

```
type(1.0)
```

```
float
```

```
isinstance(0.33333, float)
```

```
True
```

Die in der letzten Lektion gelernten Rechenoperationen funktionieren sowohl bei Floats als auch bei Ints. Werden sowohl Floats als auch Ints im selben mathematischen Ausdruck verwendet, ist das Ergebnis ein Float. Ein Float kann mit der Funktion `int()` in ein Integer umgewandelt werden. Mit der Funktion `float()` kann ein Integer in einen Float umgewandelt werden.

```
5 + 1.0 # Int + Float = Float
```

```
6.0
```

```
int(6.0)
```

```
6
```

```
float(6)
```

```
6.0
```

Floats können auch einige spezielle Werte annehmen: `Inf`, `-Inf` und `NaN`. `Inf` und `-Inf` stehen für Unendlichkeit und negative Unendlichkeit, und `NaN` steht für “not a number” und wird manchmal als Platzhalter für fehlende oder fehlerhafte numerische Werte verwendet.

```
type(float("Inf"))
```

```
float
```

```
type(float("NaN"))
```

```
float
```

Anmerkung: Python enthält einen dritten, seltenen numerischen Datentyp “complex”, der zur Speicherung von komplexen Zahlen verwendet wird.

Booleans

Booleans (“bools”) repräsentieren in Python Wahrheitswerte, die entweder True oder False sein können (siehe Wiki). Diese Werte können direkt zugewiesen oder durch den Ausgang logischer Ausdrücke und Vergleiche erzeugt werden. In Python beginnen booleans mit einem Großbuchstaben, sodass `True` und `False` als bools erkannt werden, “true” und “false” jedoch nicht. Ein Beispiel für booleans wurde bereits gesehen, als die Funktion isinstance() oben verwendet wurde.

```
type(True)
```

```
bool
```

```
isinstance(False, bool)  # Überprüfung, ob False vom Typ bool ist
```

```
True
```

Boolesche Werte können mit logischen Ausdrücken erzeugt werden. Python unterstützt alle standardmäßigen logischen Operatoren, die man erwarten würde:

```
20 > 10 # > für größer als
```

```
True
```

```
20 < 5 # < für kleiner als
```

```
False
```

```
20 >= 20 # >= und <= für größer oder gleich und kleiner oder gleich
```

```
True
```

```
10 == 10 # Verwendung von == (zwei aufeinanderfolgende Gleichheitszeichen), um
auf Gleichheit zu überprüfen
```

```
True
```

```
40 == 40.0  # Äquivalente ints und floats gelten als gleich
```

```
True
```

```
1 != 2 # != um auf Ungleichheit zu überprüfen; "nicht gleich"
```

```
True
```

```
not False # Das Schlüsselwort "not" für die Negation verwenden
```

```
True
```

```
(2 > 1) and (10 > 11) # "and" für logisches Und verwenden
```

```
False
```

```
(2 > 1) or (10 > 11) # "or" für logisches Oder verwenden
```

```
True
```

Ähnlich wie bei mathematischen Ausdrücken haben logische Ausdrücke eine festgelegte Reihenfolge der Operationen. In einer logischen Aussage werden Vergleiche wie `>`, `<` und `==` zuerst ausgeführt, gefolgt von `not`, dann `and` und schließlich `or` (siehe Details). Ein logisches Und (`and`) bewertet zwei Aussagen als wahr, wenn beide wahr sind,

während ein logisches Oder (`or`) wahr ist, wenn mindestens eine der beiden Aussagen wahr ist.

Klammern werden verwendet, um die gewünschte Reihenfolge der Operationen zu erzwingen.

```
2 > 1 or 10 < 8 and not True
```

```
True
```

```
((2 > 1) or (10 < 8)) and (not True)
```

```
False
```

Zahlen können mit der Funktion `bool()` in boolesche Werte umgewandelt werden. Alle Zahlen außer 0 werden zu True konvertiert:

```
bool(1)
```

```
True
```

```
bool(0)
```

```
False
```

Strings

Textdaten in Python werden als String oder `str` bezeichnet. Um einen String zu erstellen, umgibt man den Text mit einfachen oder doppelten Anführungszeichen:

```
type("cat")
```

```
str
```

```
type('1')
```

```
str
```

Zwei Anführungszeichen direkt nebeneinander (wie `''` oder `""`) ohne etwas dazwischen werden als der leere String bezeichnet. Der leere String stellt oft einen fehlenden Textwert dar.

Während numerische Daten und logische Daten im Allgemeinen gut handhabbar sind, können Textdaten sehr unordentlich und schwer zu bearbeiten sein. Das Säubern von Textdaten ist oft einer der mühsamsten Schritte bei der Vorbereitung von echten Datensätzen für die Analyse. In zukünftigen Lektionen wird erneut auf Strings und Funktionen eingegangen, um beim Säubern von Textdaten zu helfen.

None

In Python stellt `None` einen speziellen Datentyp dar, der häufig verwendet wird, um einen fehlenden Wert zu repräsentieren. Beispielsweise wird, wenn eine Funktion definiert wird, die nichts zurückgibt (keinen Ergebniswert liefert), standardmäßig `None` zurückgegeben.

```
type(None)
```

```
NoneType
```