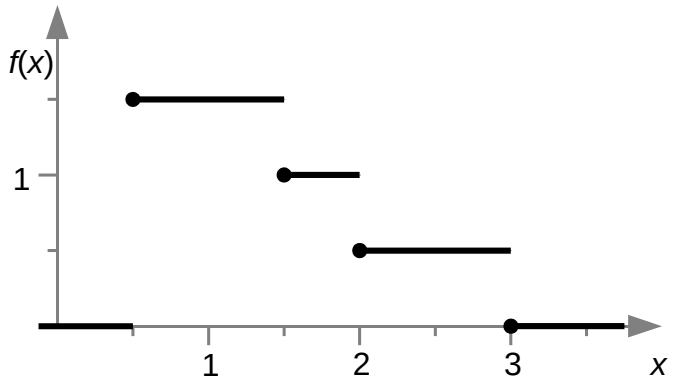
1 Stückweise konstante Funktionen (ca. 4+2+4=10 Punkte)

In dieser Aufgabe soll eine Klasse `Funktion` zur Darstellung stückweise konstanter Funktionen geschrieben werden. Hier ist ein Beispiel für so eine Funktion:



Die Funktionen werden beschrieben durch eine Menge aneinandergrenzender Intervalle, auf denen wir Funktionswerte vorgeben. Im Einzelnen soll die Klasse folgende Attribute bekommen:

- Der linke Endpunkt des ersten Intervalls (im Beispiel `0.5`).
- Eine Liste für die Länge der Intervalle (in unserem Beispiel mit Sprungstellen bei 1/2, 3/2, 3 und 3 wäre das `b=[1., 0.5, 1.]`).
- Eine Liste mit den Funktionswerten auf den Intervallen (im Beispiel wäre das `h=[1.5, 1., 0.5]`).

Außerhalb des so definierten Bereichs soll die Funktion 0 sein. Ob Ihre Funktionen rechtsseitig (wie im Bild angedeutet) oder linksseitig stetig werden, ist egal.
Schreiben Sie eine Klassendefinition mit einem Konstruktor, so dass die Beispielfunktion mittels `f = Funktion(0.5, b, h)` (`b` und `h` wie oben) instanziiert werden kann. Ihr Konstruktor soll von den Listen Kopien speichern; er darf davon ausgehen, dass alle Parameter vernünftige Werte enthalten.

```
class Funktion(object):
  def __init__(self, x0, breite, hoehe):
    self.x0 = float(x0)
    self.breite = [float(bi) for bi in breite]
    self.hoehe = list(hoehe)
```

Schreiben Sie nun noch eine Methode `integral` (die wir uns zur Klassendefinition von `Funktion` dazudenken), die für eine stückweise konstante Funktion f das Integral

$$\int_{-\infty}^{\infty} f(x)\,dx$$

als Funktionsergebnis zurückliefert. Für das oben instanziierte Objekt wäre `f.integral()` also `2.5` (weil 3/2 + 1/2 + 1/2 = 5/2).

```
def integral(self):
  sum = 0.
  for i in range(0, len(self.breite)):
    sum = sum + self.breite[i]*self.hoehe[i]
  return sum
```

Und nun noch eine Methode `wert`, die die Funktion an einer Stelle x auswertet. Im Beispiel wäre `f.wert(1.)` also `1.5` und `f.wert(5.)` wäre `0.`

```
def wert(self, x):
  if x<self.x0:
    return 0.
  xi = self.x0
  for i in range(0, len(self.breite)):
    xi = xi + self.breite[i]
    if x < xi:
      return self.hoehe[i]
  return 0.
```

2 Primfaktorzerlegung (ca. 2+3+4+6=15 Punkte)

In dieser Aufgabe soll eine Klasse zur Primfaktorzerlegung erstellt werden. Dies geschieht in mehreren Schritten.
Definieren Sie zunächst die Klasse `Primfaktoren` mit einem Konstruktor, der eine Liste als Klassenvariable `primzahlen` anlegt, in der später Primzahlen gespeichert werden sollen. Außerdem soll die kleinste Primzahl 2 gleich in der Liste gespeichert werden (Das vereinfacht den Code in einer der folgenden Aufgaben).

```
class Primfaktoren:
  def __init__(self):
    self.primzahlen = [2]
```

Erweitern Sie die Klasse um eine Methode `enthaeltFaktor`, mit der übergrüft werden soll, ob die als Parameter an die Methode übergebene Zahl durch eine der gespeicherten Primzahlen teilbar ist. In diesem Fall gibt die Methode `True`, sonst `False`. Sofern zwischen der übergebenen Zahl und der größten Primzahl aus der gespeicherten Liste keine weitere Primzahl liegt, kann mit dieser Methode bestimmt werden, ob die Zahl eine Primzahl ist. *Hinweis: Der Modulo-Operator `%` kann nützlich sein.*

```
def enthaeltFaktor(self, zahl):
  for primzahl in self.primzahlen:
    if zahl%primzahl == 0:
      return True
  return False
```

Die Klassenvariable `primzahlen` enthält eine Liste von Primzahlen (in ansteigender Reihenfolge). Mit der nächsten Methode `naechsteZahl` soll die nächstgrößere Primzahl gefunden werden und sowohl an die Liste angehängt als auch zurückgegeben werden. *Hinweis: Starten sie bei der Zahl, die eins größer ist als die letzte und prüfen Sie mit der Methoden* `enthaeltFaktor`*, ob es eine Primzahl ist. Falls nicht, nächstgrößere Zahl prüfen ...*

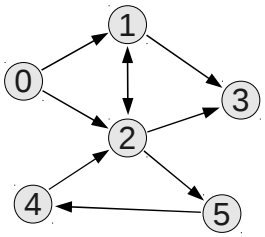
```
def naechsteZahl(self):
  zahl = self.primzahlen[-1]+1
  while True:
    noprim = self.enthaeltFaktor(zahl)
    if noprim:
      zahl = zahl+1
    else:
      self.primzahlen.append(zahl)
      return zahl
```

Die letzte Methode `primfaktoren` soll nun die eigentliche Primfaktorzerlegung einer Zahl (als Parameter an die Methode übergeben) mit Hilfe der Methode `naechsteZahl` durchführen. Eine Liste mit allen Primfaktoren (mehrfache sind auch mehrfach in der Liste) wird zurückgegeben. *Hinweis: Die übergebene Zahl wird so lange durch die erste Primzahl geteilt, wie sie ohne Rest teilbar ist, dann wird die nächste Primzahl ermittelt und durch diese geteilt, ...*

```
def primfaktoren(self, zahl):
  self.primzahlen = [2]
  primfaktoren = []
  faktor = 2
  while zahl > 1:
    if zahl%faktor == 0:
      primfaktoren.append(faktor)
      zahl /= faktor
    else:
      faktor = self.naechsteZahl()
  return primfaktoren
```

3 Graphen (ca. 7 Punkte)

Ein gerichteter Graph sei in einer Adjazenzmatrix, die durch ein numpy-Array realisiert ist, abgespeichert. Der Datentyp der einzelnen Einträge ist `bool`, eine `0`/`True` bei Index i,j bedeutet, dass eine Kante vom Knoten i zum Knoten j zeigt. Die folgende Abbildung veranschaulich dies:




```
graph = array([[1,1,1,0,0,0],
               [0,1,1,1,0,0],
               [0,1,1,1,0,1],
               [0,0,0,1,0,0],
               [0,0,1,0,1,0],
               [0,0,0,0,1,1]], dtype=bool)
```

Schreiben Sie eine Funktion `pfadSuche` mit den drei Parametern `graph` (Typ `array`), `start` (Type `int`) und `ziel` (Typ `int`), die `True` zurückgibt, wenn es einen Pfad vom Knoten `start` zum Knoten `ziel` gibt. Die Effizienz des Algorithmus ist nicht relevant.

```
def pfadSuche(graph, start, ziel):
  for i in range(graph.shape[0]):
    for j in range(graph.shape[0]):
      graph[start] = graph[start] | \
                     (graph[start][j] & graph[j])
  return graph[start][ziel]
```