In [ ]:
```
# Initialize Otter
import otter
grader = otter.Notebook("CT-02-Pyramiden.ipynb")
```

1 1 Aufwärmen

Aufgabe 1 (Listen). Erstellen Sie auf unterschiedliche Weise eine Liste `small_numbers`, welche die Zahlen 0 bis 9 enthält (beides inklusive).

In [2]:
```
# BEGIN SOLUTION
small_numbers = [0,1,2,3,4,5,6,7,8,9]
print(small_numbers)

small_numbers = list(range(10))
print(small_numbers)

small_numbers = []
for i in range(10):
    small_numbers.append(i)
print(small_numbers)
# END SOLUTION
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [ ]:
```
grader.check("q1")
```

Aufgabe 2 (Listen). Erstellen Sie eine Liste `pyramid_list` der Länge 10, die an der `i`-ten Position eine Liste der Zahlen `[0,1,...,i]` enthält. Das Ergebnis ist eine Liste aus Listen.

In [28]:
```
pyramid_list = [list(range(i+1)) for i in range(10)] # SOLUTION
# BEGIN SOLUTION
pyramid_list = []
for i in range(10):
    pyramid_list.append([])
    for j in range(i+1):
        pyramid_list[i].append(j)
# END SOLUTION
pyramid_list
```

```
Out[28]: [[0],
          [0, 1],
          [0, 1, 2],
          [0, 1, 2, 3],
          [0, 1, 2, 3, 4],
          [0, 1, 2, 3, 4, 5],
          [0, 1, 2, 3, 4, 5, 6],
          [0, 1, 2, 3, 4, 5, 6, 7],
          [0, 1, 2, 3, 4, 5, 6, 7, 8],
          [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]]
```

```
In [ ]: grader.check("q2")
```

Aufgabe 3 (Zeichenketten). Wir bezeichnen eine Zeichenfolge Palindrom wenn sie vorwärts und rückwärts gelesen das gleiche Wort ergibt. Z.B. ist `anna` oder auch `aabaa` ein Palindrom. Schreiben Sie einen Code der für eine beliebige Zeichenkette `text` prüft ob es sich um ein Palimdrom handelt. Testen Sie Ihren Code.

```
In [34]: text = 'anna'
         # BEGIN SOLUTION
         palindrome = True
         for i in range(len(text)//2):
             palindrome = palindrome and text[i] == text[-(i+1)]
         print(palindrome)

         text == text[::-1]
         # END SOLUTION
```

```
True
```

```
Out[34]: True
```

Aufgabe 4 (Funktionen). Schreiben Sie eine Funktion `is_even(n)`, die Ihnen wahr `True` zurückgibt wenn `n` eine gerade Zahl ist und sonst `False`. Gehen Sie davon aus, dass `n` eine ganze Zahl ist.

```
In [2]: def is_even(n):
            return n % 2 == 0 # SOLUTION
```

```
In [ ]: grader.check("q4")
```

Aufgabe 5 (Funktionen, Kontrollstruturen). Schreiben Sie eine Funktion `specify_number(n)`, die einen Text ausgibt, der besagt ob die ganze Zahl `n` gerade oder ungerade und positiv oder negativ ist (0 werten wir als positiv).

```
In [139]: def specify_number(n):
              # BEGIN SOLUTION
              if is_even(n) and n >= 0:
                  print(f'{n} is even and positive')
              elif not is_even(n) and n >= 0:
                  print(f'{n} is not even and positive')
              elif is_even(n) and n < 0:
                  print(f'{n} is even and negative')
              else:
                  print(f'{n} is not even and negative')
              # END SOLUTION
```

Aufgabe 6. (Kontrollstrukturen) Geben Sie 20 Mal `'Die richtige Antwort ist 42!'` aus.

```
In [ ]: # BEGIN SOLUTION
        for _ in range(20):
            print('Die richtige Antwort ist 42!')
        # END SOLUTION
```

Aufgabe 7 (Listen, Kontrollstrukturen). Erstellen Sie eine Liste `numbers`, die alle ungeraden positiven Zahlen, welche durch 7 teilbar und kleiner als 1000 sind.

```
In [38]: numbers = [i for i in range(1000) if not is_even(i) and i % 7 == 0] # SOLUTION
         # BEGIN SOLUTION
         numbers = []
         for i in range(1000):
             if not is_even(i) and i % 7 == 0:
                 numbers.append(i)
         # END SOLUTION
```

```
In [ ]: grader.check("q7")
```

Aufgabe 8 (Funktionen, Kontrollstrukturen). Schreiben Sie eine Funktion `sum_squares(n)`, die Ihnen die Summe aller Quadratzahlen von 1 bis einschließlich `n` berechnet und zurückgibt. Gehen Sie davon aus, dass `n` eine natürliche Zahl $\mathbb{N}$ (0 eingeschlossen) ist.

```
In [125]: def sum_squares(n):
              # BEGIN SOLUTION
              result = 0
              for i in range(n):
                  result += (i+1)**2
              return result
              # END SOLUTION
```

```
In [ ]: grader.check("q8")
```

---

2 2 Münzliebhaber*in

Sie möchten heute gerne einen bestimmten Geldbetrag `money` in Münzen `coins` umwandeln. Doch zuvor möchten Sie wissen wie viele Münzen Sie denn minimal benötigen, schließlich möchten Sie sich nicht zu Tode Schleppen.

Unsere Währung ist der Euro. Es gibt 2 und 1 Euro Münzen (200, 100 Cent) sowie 50, 20, 10, 5, 2 und 1 Cent Münzen. Sie brauchen die wenigsten Münzen für einen bestimmten Geldbetrag wenn Sie die Anzahl der wertvollsten Münzen maximieren. Warum?

Aufgabe 9 (Funktionen, Kontrollstrukturen). Schreiben Sie eine Funktion `count_coins(money)`, die Ihnen die Anzahl der Münzen, die Sie **minimal** benötigen, berechnet! Dabei soll `money` der Geldbetrag in Euro sein, welchen Sie in Münzen umwandeln möchten. Gehen Sie davon aus, dass `money` eine positive Fließkommazahl `float` ist und runden Sie (mit `round()`) auf den nähersten Geldbetrag, z.B. `100.2356` wird auf `100.24` gerundet und `50.224` auf `50.22`.

```
In [146]: def count_coins(money):
              # BEGIN SOLUTION
              cents = cents = round(money * 100)
              count = 0
              while cents > 0:
                  if cents >= 200:
                      cents = cents - 200
                  elif cents >= 100:
                      cents = cents - 100
                  elif cents >= 50:
                      cents = cents - 50
                  elif cents >= 20:
                      cents = cents - 20
                  elif cents >= 10:
                      cents = cents - 10
                  elif cents >= 5:
                      cents = cents - 5
                  elif cents >= 2:
                      cents = cents - 2
                  else:
                      cents = cents - 1
                  count = count + 1
```

```
        return count

    # or much more compact
    def count_coins_short(money):
        coins = [200, 100, 50, 20, 10, 5, 2, 1]
        count = 0
        cents = round(money * 100)
        while cents > 0:
            for coin in coins:
                if cents >= coin:
                    cents = cents - coin
                    break
            count += 1
        return count

    # or even much more efficient using math (modulo division)
    def count_coins_math(money):
        coins = [200, 100, 50, 20, 10, 5, 2, 1]
        count = 0
        cents = round(money * 100)
        for coin in coins:
            # cents = coin * k + r (k is a natural number)
            k = cents // coin
            r = cents % coin
            cents = cents - k * coin
            count += k
        return count
    # END SOLUTION
```

```
In [ ]: grader.check("q9")
```

---

3 3 Pyramidenbau (schwer)

3.1 3.1 Einleitung

Sie kennen vielleicht noch die alten sogenannten Jump & Run Konsolenspiele wie etwa Mario, Sonic und viele mehr.

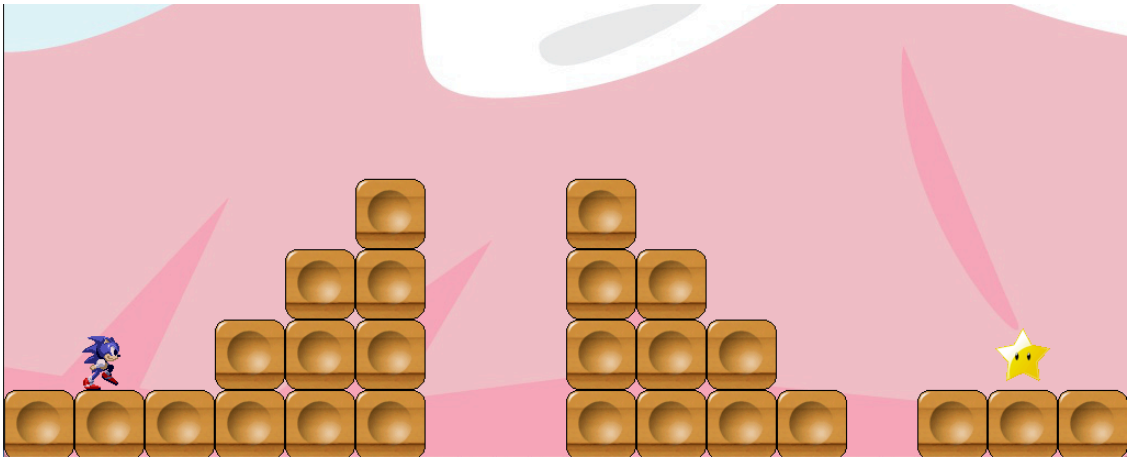
Das Spielprinzip ist einfach: In einer zweidimensionalen Welt, in der sich die Spielfigur nur nach vorne bzw. zurück und hoch bzw. runter bewegen kann, muss diese Figur ein bestimmtes (örtliches) Ziel erreichen. Auf dem Weg lauern Gefahren und Ihre Figur muss Hindernisse überspringen. Sie durchlaufen mehrere Levels, deren Schwierigkeitsgrad ansteigt.

Interessanterweise gibt es diese Spiele heute immernoch - sie haben einen gewissen Kultstatus. Besonders begehrt sind Spiele bei denen Spieler*innen ihre eigenen Welten erstellen können und diese dann von anderen Spieler*innen gespielt und bewertet werden.

Ist die Spielwelt aus einfachen Elementen welche in einem regulären Gitter angeordnet sind aufgebaut, so kann eine solche Welt als lesbarer Text in eine Datei geschrieben werden. Ein Beispiel: Stellen Sie sich vor, folgender Text stünde in einer Textdatei:

```
     #  #
    ##  ##
 P ###  ###   *
######  #### ###
```

Dabei könnten wir definieren, dass `P` die Startposition der Spielfigur ist, `#` quadratische Bauklötze und `*` das zu erreichende Ziel ist. Diese Information könnte von einem Programm in eine entsprechende Welt umgewandelt werden.



Auch hierbei handelt es sich um eine **Interpretation**, eine ganze Spielewelt durch eine Folge von Zeichen **repräsentiert**!

3.2 3.2 Aufgabenbeschreibung

Ihr Aufgabe ist es ein `Python`-Programm zu schreiben, welches Ihnen eine Pyramide der Form

```
   #  #
  ##  ##
 ###  ###
####  ####
```

erzeugt, wobei die Höhe der Pyramide als Eingabeargument übergeben werden und zwischen 1 und 8 liegen soll.

Aufgabe 10 (Funktionen, Kontrollstrukturen). Entwickeln Sie eine Funktion `pyramid(height)` die die oben beschriebene Pyramide der Höhe `height` als Zeichenkette `str` erzeugt und zurückgibt. Gehen Sie davon aus, dass `height` eine nicht zu große natürliche Zahl (0 ausgeschlossen) ist.

Zeichenketten können Sie mit dem `+`-Operator verketten. Eine neue Zeile können Sie mit der Zeichenkette `'\n'` generieren. Zum Beispiel:

```
In [106]: print('abcd' + 'efg' + '\n' + 'hijk')
```

```
abcdefg
hijk
```

Oft wollen wir überflüssig Zeilen am Ende einer Zeichenkette (oder auch Textdatei) vermeiden.

```
In [109]: print('abcd' + 'efg' + '\n' + 'hijk\n\n\n')
```

```
abcdefg
hijk
```

```
In [1]: def pyramid(height):
            # BEGIN SOLUTION
            gap = 2
            width = height*2 + gap
            pyramid = ''
            for i in range(1,height+1,1):
                empty = (width-gap-2*i) // 2
                for _ in range(empty):
                    pyramid += ' '
                for _ in range(i):
                    pyramid += '#'
                for _ in range(gap):
                    pyramid += ' '
```

```
            for _ in range(i):
                pyramid += '#'
            for _ in range(empty):
                pyramid += ' '
            if i < height:
                pyramid += '\n'
        return pyramid

    def pyramid(height):
        # BEGIN SOLUTION
        gap = 2
        width = height*2 + gap
        pyramid = ''
        for i in range(1,height+1,1):
            empty = (width-gap-2*i) // 2
            tiles = i * '#'

            pyramid += empty * ' '
            pyramid += tiles
            pyramid += gap * ' '
            pyramid += tiles

            if i < height:
                pyramid += '\n'
        return pyramid

    # END SOLUTION
```

```
In [ ]: grader.check("q10")
```

Aufgabe 11 (Funktionen, Kontrollstrukturen). Entwickeln Sie eine Funktion `print_pyramid()` die nach einer Eingabe (der Höhe) frägt und Ihnen die oben beschriebene Pyramide der Höhe dieser Eingabe erzeugt genauer gesagt als ausgibt. Verwenden Sie Ihre Funktion `pyramid(height)`. Bei fehlerhafter Eingabe sollte stattdessen eine Fehlermeldung ausgegeben werden. Probieren Sie Ihre Funktion aus.

Um einen Wert in einem Notebook einzulesen, können Sie die *built-in*-Funktion `input()` verwenden, zum Beispiel:

```
In [140]: height = input('height = ')
          print('data type:' + str(type(height)))
          print('value: ' + str(height))
```

```
height = 7
data type:<class 'str'>
value: 7
```

Mit der folgenden Funktion können Sie testen ob eine Zeichenkette tatsächlich eine natürliche Zahl $\mathbb{N}$ (0 ausgeschlossen) ist.

```
In [11]: def is_natural_number(value):
             try:
                 return int(value) > 0 and int(value) == value
             except ValueError:
                 return False
```

```
In [111]: def print_pyramid():
              # BEGIN SOLUTION
              height = input('height = ')
              if is_natural_number(height):
                  height = int(height)
                  if 1 <= height <= 8:
                      print(pyramid(height))
                  else:
                      print(str(height) + ' is not within [1;8]')
              else:
                  print(height + ' is not a whole positive number!')
              # END SOLUTION
```

```
In [77]: print_pyramid()
```

```
height = 6
     #  #
    ##  ##
   ###  ###
  ####  ####
 #####  #####
######  ######
```

To double-check your work, the cell below will rerun all of the autograder tests.

```
In [ ]: grader.check_all()
```

3.3 Submission

Make sure you have run all cells in your notebook in order before running the cell below, so that all images/graphs appear in the output. The cell below will generate a zip file for you to submit.

```
In [ ]: grader.export(pdf=False, force_save=True)
```