Übungsblatt 02

Aufgabe 1

Definiere eine Funktion `vergleiche(x,y,genau)`, die die beiden Fließkommazahlen `x` und `y` vergleicht und beurteilt, ob diese innerhalb des Toleranzparameters `genau` gleich sind. Sie soll einen boolschen Funktionswert zurückgeben:

$$vergleiche(x, y, genau) = \begin{cases} True & falls \quad |x - y| < genau \\ False & \text{sonst} \end{cases}$$

Im Hauptprogramm soll `vergleiche(...)` mit vordefinierten Zahlen aufgerufen werden und über `print()` eine Ausgabe erfolgen. z.B.:

- für die Parameter (1, 1.1, 0.2):

```
Näherungsweise gleich
```

- für die Parameter (0.9, 1.3, 0.1):

```
NICHT näherungsweise gleich
```

In [1]:
```python
# TODO Aufgabe 1a) Klassische Funktionsdeklaration
def vergleiche(x,y,genau) :
    return abs(x-y) < genau

# Hauptprogramm
if vergleiche(1,1.1,0.2) :
    print("Näherungsweise gleich")
else :
    print("NICHT Näherungsweise gleich")

if vergleiche(0.9, 1.3, 0.1) :
    print("Näherungsweise gleich")
else :
    print("NICHT Näherungsweise gleich")
```

```
Näherungsweise gleich
NICHT Näherungsweise gleich
```

In [2]:
```python
# TODO Aufgabe 1b) mittels des Lambda-Formats
vergleiche = lambda x, y, genau: True if abs(x-y) < genau else False

# Hauptprogramm
if vergleiche(1,1.1,0.2) :
    print("Näherungsweise gleich")
else :
    print("NICHT Näherungsweise gleich")

if vergleiche(0.9, 1.3, 0.1) :
    print("Näherungsweise gleich")
else :
    print("NICHT Näherungsweise gleich")
```

```
Näherungsweise gleich
NICHT Näherungsweise gleich
```

Aufgabe 2

Definiere eine Funktion `initialen(...)`, die mit einer **beliebigen** Anzahl an Stringparametern aufgerufen wird und deren Wert ein String mit den Initialen der übergebenen Person sind.

Beispiele:
initialen("Helmut", "Kohl") → "HK"
initialen("Hugo", "Egon", "Balder") → "HEB"
initialen("Johan", "Riley", "Fyodor", "Taiwo", "Samuel", "Klum") → "JRFTSK"

In [3]:
```python
# TODO Aufgabe 2
def initialen(*namen) :
    erg = ""
    for i in namen :
        if len(i) != 0 :
            erg += i[0]
    return erg

# Hauptprogramm
print(initialen(""))
print(initialen("Helmut", "Kohl"))
print(initialen("Hugo", "Egon", "Balder"))
print(initialen("Johan", "Riley", "Fyodor", "Taiwo", "Samuel", "Klum"))
```

```
HK
HEB
JRFTSK
```

Aufgabe 3

Ein früher verwendetes Verfahren zur Datenkomprimierung besteht darin, nur die 15 häufigsten Zeichen zu senden und alle anderen zu unterdrücken. Diese sind:

A, C, D, E, F, H, I, L, N, O , R, S, T, U, Leerzeichen

Definiere eine Funktion `komprimiere(text)`, die einen in Großbuchstaben übergebenen String nach diesem Schema verkürzt und als Funktionswert zurückgibt.
Beispiel: BIERGARTEN UM ACHT → IERARTEN U ACHT

In [4]:
```python
# TODO Aufgabe 3
def komprimiere(text) :
    erg = ""
    for i in text :
        if i in "ACDEFHILNORSTU " :
            erg += i
    return erg

# Hauptprogramm
print(komprimiere("BIERGARTEN UM ACHT"),"<-- Funktionswert")
print("IERARTEN U ACHT","<-- Sollwert")
```

```
IERARTEN U ACHT <-- Funktionswert
IERARTEN U ACHT <-- Sollwert
```

Aufgabe 4

Bekannt aus dem letzten Übungsblatt:

Zur Berechnung von $\sqrt{a}$ kann die folgende Zahlenfolge herangezogen werden:

$$x_{n+1} = \frac{1}{2}(x_n + \frac{a}{x_n})$$

mit

$$x_0 = \frac{a+1}{2}$$

Schreiben Sie dieses Mal eine **rekursive** Funktion `wurzel(...)` mit zwei Argumenten:

- `a` : ist die Zahl, deren Wurzel berechnet werden soll (notwendige Angabe)
- `n` : ist die Anzahl der rekursiven Aufrufe (optional, Default = 10)

Beispielaufrufe:

```
>>> wurzel(2, 3)
1.4142156862745097
>>> wurzel(2)
1.414213562373095
```

In [4]:
```
# TODO Aufgabe 4
def wurzel(a, n=10):
    if n == 1:
        return 0.5*(a+1)
    else:
        return 0.5*(wurzel(a,n-1)+float(a)/wurzel(a,n-1))

print (wurzel(2,3))
print (wurzel(2))
```

```
1.4142156862745097
1.414213562373095
```

Zusatzaufgabe zu Aufgabe 4:

Wenn Sie `wurzel(2,25)` aufrufen, wie lange dauert die Ausführung? Wie lange würde dann wohl `wurzel(2,2700)` dauern? (Nicht ausprobieren, wenn `wurzel(2,25)` schon mehrere Sekunden gedauert hat!)

Überlegen Sie, woran das liegen könnte und versuchen Sie, die rekursive Funktion soweit zu optimieren, dass Sie möglichst die gesamte Rekursionstiefe ausnutzen können, sodass auch ein Aufruf von `wurzel(2,2700)` nur wenige Millisekunden dauert.

In [2]:
```
# Optimierte wurzel-Funktion
def wurzel(a, n=10):
    if n == 1:
```

```
        return 0.5*(a+1)
    else:
        vorigeNaeherung = wurzel(a,n-1)     # nur ein rekursiver Aufruf
        return 0.5*(vorigeNaeherung+float(a)/vorigeNaeherung)

print (wurzel(2,2970)) # bei n > 2970 -> RecursionError: maximum recursion depth
```

```
1.414213562373095
```