Informatik Q1 Abels

Rekursion vs. Iteration

Test.py

```python
def function(n):
    s = 0
    for i in range(n+1):
        s += i
    return s

print(function(5))
```

Terminal

```
? ? ?
```

Iteration

Iteration.py

```
1  def sum(n):
2      s = 0
3      for i in range(n+1):
4          s += i
5      return s
```

Eine Folge von Anweisungen wird schrittweise in einer Schleife (z. B. for-Schleife) abgearbeitet.

Verbraucht i. A. weniger Ressourcen.

Bei fehlerhafter Abbruchbedingung kann es zur Endlosschleife kommen.

Test.py

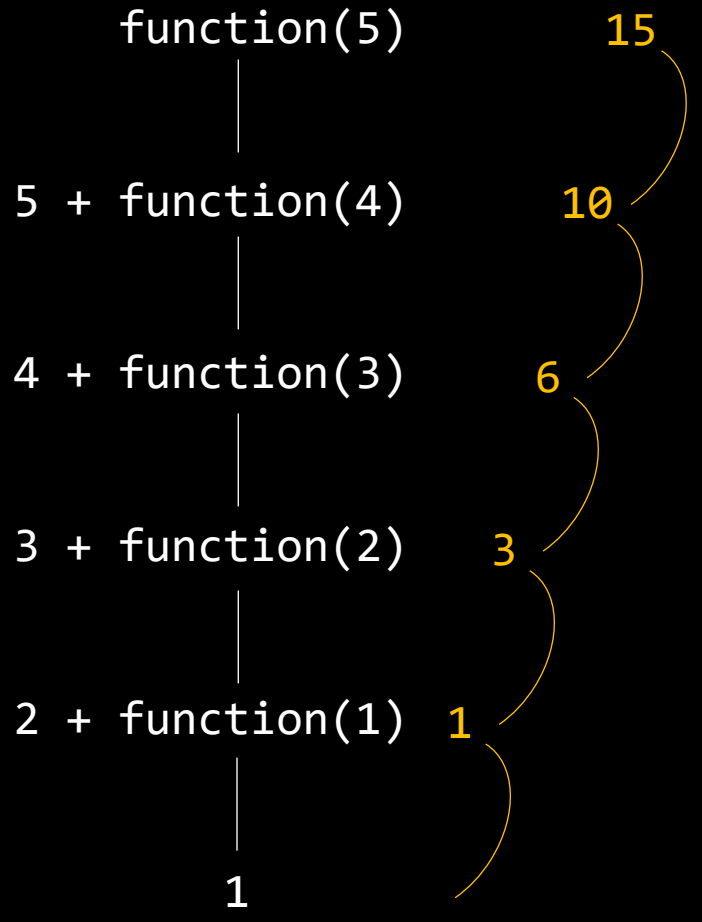
```python
def function(n):
    if n == 1:
        return 1
    return n + function(n-1)

print(function(5))
```

Terminal

```
? ? ?
```

Test.py

```python
def function(n):
    if n == 1:
        return 1
    return n + function(n-1)

print(function(5))
```



Rekursion

Eine Funktion ruft sich zur Lösung eines Problems selbst auf.

Sie folgt dem “Teile und Herrsche”-Prinzip.

Rekursion.py

```
1  def sum(n):
2      if n == 1:
3          return 1
4      return n + sum(n-1)
```

Sie bricht bei der Lösung des kleinsten Problems ab. Man spricht von der sog. Abbruchbedingung.

Der Code wird i. A. kürzer und ist leichter nachvollziehbar.

Jede Rekursion lässt sich auch durch eine Iteration ersetzen!

Übung 1

Schreibe eine Klasse **Fakultaet**:

Fakultaet.py

```python
def fak_iterative(n):
    # DEIN CODE

def fak_recursive(n):
    # DEIN CODE

print(f"iterativ: 5! = {fak_iterative(5)}")
print(f"rekursiv: 5! = {fak_recursive(5)}")
```

Übung 1

Fakultaet.py

```python
def fak_iterative(n):
    if n == 0 or n == 1:
        return 1
    product = 1
    for i in range(2, n+1):
        product *= i
    return product

def fak_recursive(n):
    if n == 0 or n == 1:
        return 1
    return n * fak_recursive(n-1)

print(f"iterativ: 5! = {fak_iterative(5)}")
print(f"rekursiv: 5! = {fak_recursive(5)}")
```

Fibonacci

Leonardo von Pisa (ca. $\mathbf{1170 - 1250}$) besser unter dem Namen Fibonacci (Sohn des Bonacci, wobei Bonacci bedeutet: „von gutem Wesen“ o. Ä.) bekannt, stellte in seinem wichtigsten Werk, dem Liber Abaci von $\mathbf{1202}$ die harmlos erscheinende Kaninchenaufgabe:

„Jemand brachte ein frisch geborenes Kaninchenpaar in einen allseits von Wänden umgebenen Ort, um herauszufinden, wie viele Paare aus diesem Paar in einem Jahr entstehen würden. Es sei die Natur der Kaninchen, pro Monat ein neues Paar hervorzubringen und im zweiten Monat nach der Geburt erstmals zu gebären. Todesfälle sollen nicht eintreten.“

1	1	2	3	5	8	…
a_0	a_1	a_2	a_3	a_4	a_5	

$$\boldsymbol{a_n = a_{n-1} + a_{n-2}}$$

Übung 2

a) Schreibe eine Klasse **`Fibonacci`**:

Fibonacci.py

```python
def fib_iterative(n):
    # DEIN CODE

def fib_recursive(n):
    # DEIN CODE

print(f"iterativ: fib(5) = {fib_iterative(5)}")
print(f"rekursiv: fib(5) = {fib_recursive(5)}")
```

b) Stelle in einer Visualisierung (Ableitungsbaum) dar, welche Funktionsaufrufe bei der rekursiven Variante für den Fall `n = 6` getätigt werden.

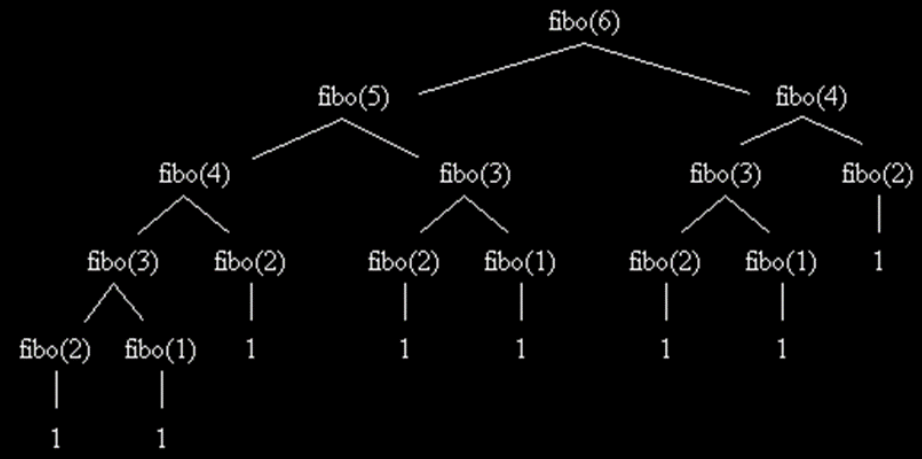
c) Gehe anhand dessen auf die Ineffizienz ein des rekursiven Algorithmus ein.

d) Erkläre folgende Unterscheidung: lineare Rekursion vs Mehrfachrekursion

Übung 2

Fibonacci.py

```python
def fib_iterative(n):
    numbers = [1, 1]
    for i in range(2, n+1):
        numbers.append(numbers[i-1] + numbers[i-2])
    return numbers[n]

def fib_recursive(n):
    if n == 0 or n == 1:
        return 1
    return fib_recursive(n-1) + fib_recursive(n-2)

print(f"iterativ: fib(5) = {fib_iterative(5)}")
print(f"rekursiv: fib(5) = {fib_recursive(5)}")
```



Reiskorn

Der Legende nach begeisterte das Schachspiel den König von Indien so sehr, dass er den Erfinder des Spiels suchen ließ. Aus Dankbarkeit wollte er einem Untertan (ein Mathematiklehrer namens Buddhiram) jeden Wunsch erfüllen. Dieser wünschte sich daraufhin ein Reiskorn auf dem ersten Feld, zwei auf dem zweiten Feld, vier auf dem dritten Feld, 8 auf dem vierten Feld usw. Der König lächelte zunächst über die Einfalt seines Untertanen, stellte jedoch bald fest, dass der Wunsch nicht zu erfüllen war.

Übung 3

a) Zur Berechnung der Reiskörner auf dem Feld **n** entwickelte Harry folgendes Programm. Wieso funktioniert sein Programm nicht?

Reiskorn.py

```python
def reis_auf_feld(n):
    return 2 * reis_auf_feld(n-1)
```

b) Schreibe eine Klasse **Reiskorn**:

Reiskorn.py

```python
def reis_auf_feld(n):
    # DEIN CODE

def reis_bis_feld(n):
    # DEIN CODE

print(f"Raus auf Feld 13: {reis_auf_feld(13)}")
print(f"Reis bis Feld 64: {reis_bis_feld(64)}")
```

Übung 3

Reiskorn.py

```python
def reis_auf_feld(n):
    if n == 1:
        return 1
    return 2 * reis_auf_feld(n-1)

def reis_bis_feld(n):
    if n == 1:
        return 1
    return reis_auf_feld(n) + reis_bis_feld(n-1)

print(f"Raus auf Feld 13: {reis_auf_feld(13)}")
print(f"Reis bis Feld 64: {reis_bis_feld(64)}")
```

Übung 4

a) Welche Funktionswerte liefert die folgende Funktion bei den Parametern ”HANNAH”, ”SARAH” und ”OTTO” ?

b) Beschreibe die Funktionalität der Funktion.

Strings.py

```python
def textprobe(s):
    if len(s) == 0 or len(s) == 1:
        return True
    return s[0] == s[-1] and textprobe(s[1:-1])
```

c) Schreibe eine Klasse **Strings** und ergänze folgende Funktion:

d) Vereinfache obige Funktion durch deine neue Funktion.

Strings.py

```python
def umdrehen(s):
    # DEIN CODE

def textprobe_neu(s):
    # DEIN CODE

print(f"HANNAH: {textprobe_neu("HANNAH")}")
print(f"SARAH: {textprobe_neu("SARAH")}")
print(f"OTTO: {textprobe_neu("OTTO")}")
```

Übung 4

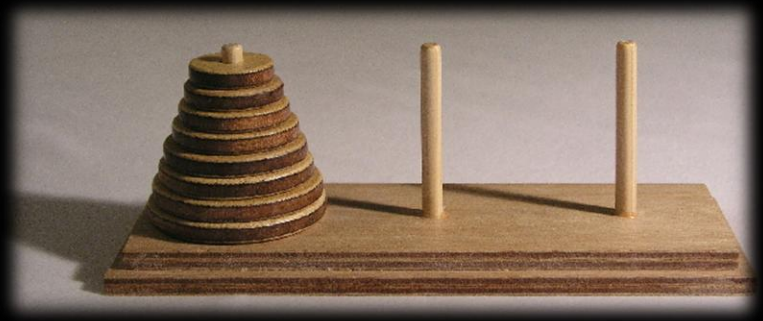
Strings.py

```python
def textprobe(s):
    if len(s) == 0 or len(s) == 1:
        return True
    return s[0] == s[-1] and textprobe(s[1:-1])

def umdrehen(s):
    if len(s) == 0 or len(s) == 1:
        return s
    return s[-1] + umdrehen(s[1:-1]) + s[0]

def textprobe_neu(s):
    return s == umdrehen(s)

print(f"HANNAH: {textprobe("HANNAH")}")
print(f"SARAH: {textprobe("SARAH")}")
print(f"OTTO: {textprobe("OTTO")}")
print(f"HANNAH: {textprobe_neu("HANNAH")}")
print(f"SARAH: {textprobe_neu("SARAH")}")
print(f"OTTO: {textprobe_neu("OTTO")}")
```

Tagebucheintrag

Rekursion vs. Iteration

Türme von Hanoi



Das Spiel wird von einer Person gespielt. Es besteht aus drei gleich großen Stäben, auf die mehrere gelochte Scheiben gesteckt werden, alle verschieden groß.

Zu Beginn liegen alle Scheiben auf dem linken Stab, der Größe nach geordnet, mit der größten Scheibe unten und der kleinsten oben (siehe Bild).

Ziel des Spiels ist es, den kompletten Scheiben-Stapel vom linken Stab auf den rechten Stab zu versetzen. Hierbei gelten zwei Regeln:

1. Es darf immer nur eine Scheibe gleichzeitig bewegt werden.
2. Die bewegte Scheibe darf nicht auf eine kleinere Scheibe gesteckt werden.

Folglich sind zu jedem Zeitpunkt des Spieles die Scheiben auf jedem Feld der Größe nach geordnet.

Wochenübung

- Probiere selbst aus: Wie viele Züge benötigst du auf jeden Fall für 3 Scheiben? Wie viele für 4? Wie viele benötigst du allgemein für n Scheiben?
- Beschreibe deine Strategie in einem Struktogramm.
- Entwickle ein Programm **Hanoi**, in dem eine rekursive Funktion implementiert ist, die die Zugfolge in der Console ausgibt.

Hanoi.py

```
1  def hanoi(n, quelle, ziel, hilfe):
2      # DEIN CODE
3
4  hanoi(3, 'A', 'C', 'B')
```