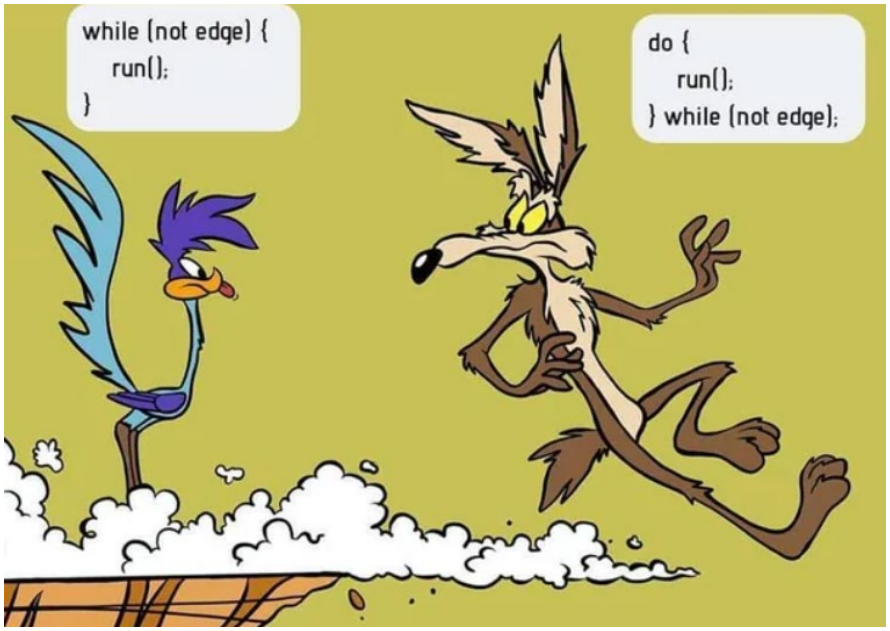
Lösungsalgorithmen & Programmieren

- Block 02 (Schleifen)-

Abend 01



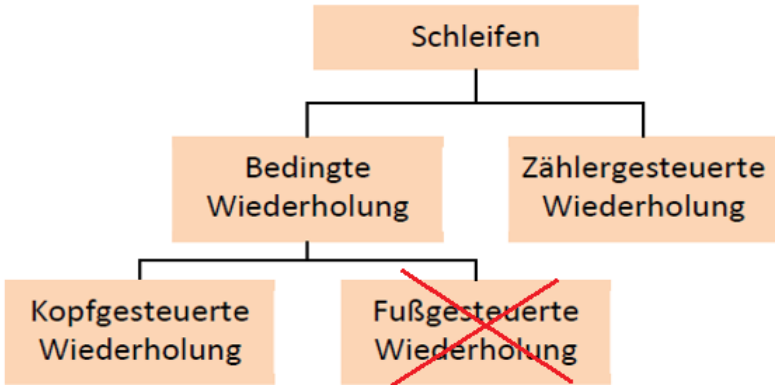
🎯 Leistungs-/Lernziele dieses Blocks:

Die Studierenden …
können Verzweigungen mit if, elif, else umsetzen (Repetition)
kennen die while Schleife und können diese in Pythonprogrammen umsetzen
kennen die Sprunganweisungen break und continue und können diese in Pythonprogrammen umsetzen
kennen den Aufbau der for-Schleife und können sie mit der Funktion Range als Zählschleife einsetzen

Inhaltsverzeichnis:

1 Schleifen

Kontrollstrukturen bestehen aus zwei Kategorien: Verzweigungen (z.B. if-else) und Schleifen (z.B. while).
Bei Verzweigungen geht es prinzipiell darum, ob eine Anweisung oder ein Block von Anweisungen ausgeführt werden soll oder nicht.
Bei Schleifen geht es darum, dass eine Anweisung oder ein Block von Anweisungen unter Umständen mehrmals ausgeführt werden soll.



Während es bei vielen andern Programmiersprachen kopf und fussgesteuerte Schleifen gibt, existieren bei Python nur kopfgesteuerte Schleifen. Eine Art do -while Schleife gibt es nicht.

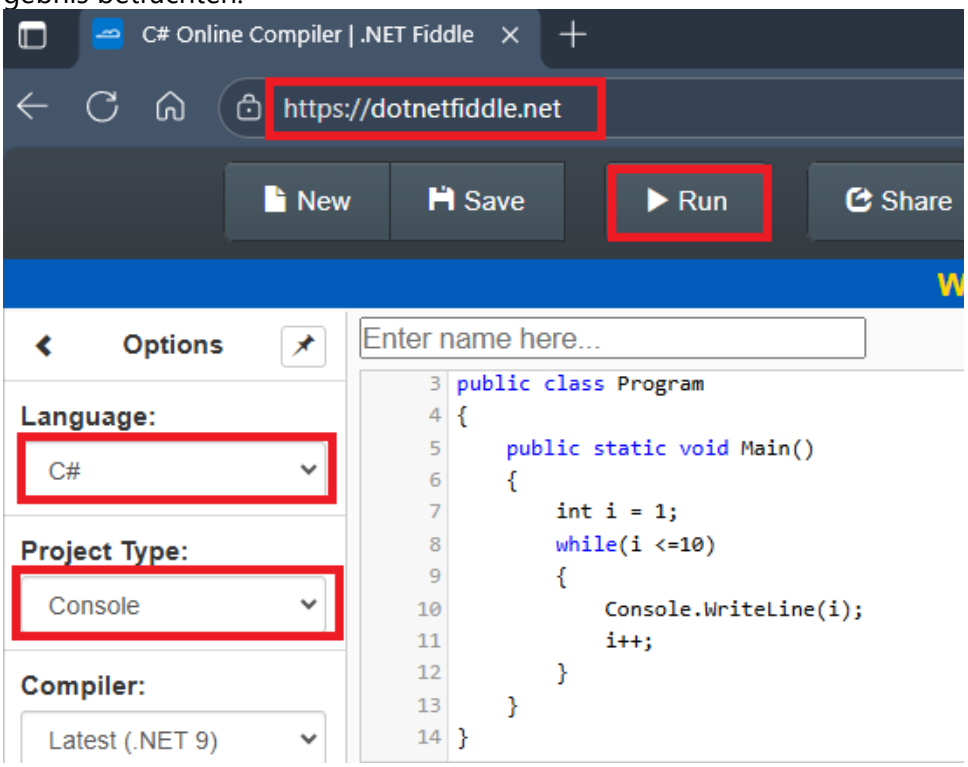
Betrachten wir die folgende Übersicht von drei verschiedenen Schleifentypen in c# / Java Code, welche alle die gleiche Ausgabe machen: die Zahlen von 1 bis 10 ausgeben:

```
int i = 1;
while(i <=10)
{
    Console.WriteLine(i);
    i++;
}
```

```
int i = 1;
do
{
    Console.WriteLine(i);
    i++;
} while (i <= 10);
```

```
for (int i = 1;  i <= 10 ; i++)
    Console.WriteLine(i);
```

Wenn sie möchten, können sie den c# Programmcode bei https://dotnetfiddle.net ausführen und das Ergebnis betrachten:



1.1 Kopfgesteuerte Wiederholung (While)

Ein Loop besteht wie eine Verzweigung aus einer Bedingung und einem Anweisungsblock. Der Anweisungsblock wird so lange wiederholt, wie die Bedingung True ist.

Pythoncode while im Allgemeinen

```
while condition:
    # Anweisungsblock
```

- Nach dem Schlüsselwort `while` steht die Bedingung.
- Ist die Bedingung `True`, wird der Anweisungsblock ausgeführt und die Bedingung erneut geprüft. Solange die Bedingung `True` ist, wird der Anweisungsblock wiederholt.
- Ist die Bedingung `False`, wird der Anweisungsblock übersprungen und das Programm nach der `while`-Anweisung fortgesetzt.

Pythoncode while konkret

```
counter = 1
while counter < 10:
    print(counter)
    counter = counter + 2
```

Was ist die Ausgabe des Programms:

1.2 Sprunganweisungen in Schleifen (Break und Continue)

Break

Mit der break-Anweisung kann ein aktuell durchlaufender Anweisungsblock abgebrochen werden. Die Schleife wird beendet.

Pythoncode mit break

```
counter = 0
while True:     # läuft endlos, da die Bedingung stets True gibt
    counter = counter + 1
    if counter >= 3:     # Wenn die Zählvariable grösser gleich 3 ist, soll die Schleife abgebrochen werden
        break
    print(counter)
```

Was ist die Ausgabe des Programms:

Continue

Mit der continue-Anweisung kann ein aktuell durchlaufender Anweisungsblock abgebrochen werden und mit den nächsten fortgefahren werden.

Pythoncode mit continue

```
counter = 0
while counter < 5:
    counter = counter + 1
    if counter == 3:     # Die Zahl 3 wird nicht ausgegeben, da continue
        continue
    print(counter)
```

Was ist die Ausgabe des Programms:

1.3 Zählschleife mit for

Für die Angabe der Anzahl Wiederholungen kann die Funktion range mit deren unterschiedlicher Schreibweise sehr gut verwendet werden.
Range liefert eine Anzahl Elemente mit einem bestimmten Wert.

Erklärung	Beispiel	Anzahl Elemente	Werte
`range(stop)`	`range(5)`	5	0, 1, 2, 3, 4
`range(start, stop)`	`range(2, 8)`	6	2, 3, 4, 5, 6, 7
`range(start, stop, step)`	`range(2, 16, 2)`	7	2, 4, 6, 8, 10, 12, 14

Wenn bereits zu Beginn der Schleife bekannt ist, wie oft der Anweisungsblock wiederholt werden soll, ist die Verwendung der for-Schleife die kompakteste Form.

Pythoncode for im Allgemeinen

```
for iterator in range:
    # Anweisungsblock
```

- Nach dem Schlüsselwort `for` steht der Iterator, welcher bei jedem Durchlauf der Schleife einen Wert des dahinterstehenden Ranges annimmt.
- Die Anzahl Elemente des Ranges bestimmen, wie oft der Anweisungsblock wiederholt wird.

Pythoncode for konkret

```
for i in range(2, 16, 2):
    print(i)
```

Was ist die Ausgabe des Programms:

📝 Projekt: Gemeinsam ein Beispiel lösen

Schwierigkeitsgrad: ⭐	⏳ Zeit: 15 Minuten	Arbeitsform: Plenum 👫

Wir werden gemeinsam ein Beispiel zur Berechnung der Fakultät erstellen.
Wir fragen den Benutzer nach einer Zahl. Wir berechnen die Fakultät dieser Zahl und geben sie aus.

Zahl n eingeben: 5
Die Fakultät von 5 ist 120

Tipp: Fakultät (auf dem Taschenrechner !) ist ein mathematischer Ausdruck, der das Produkt aller ganzen Zahlen von 1 bis n beschreibt: $n! = 1 \times 2 \times 3 \times 4 \times \cdots \times n$

Beispiele (und Testwerte):
0! = 1
1! = 1
2! = 1x2 = 2
3! = 1x2x3 = 6
4! = 1x2x3x4 = 24
5! = 1x2x3x4x5 = 120
etc.

Wofür braucht man Fakultät?

- In der Kombinatorik: z. B. „Wie viele Möglichkeiten gibt es, 5 Objekte anzuordnen?“ → 5! = 120
- In der Wahrscheinlichkeitsrechnung (z. B. Lotto, Permutationen)
- In der Informatik & Mathematik (z. B. Rekursion, Algorithmen, Wachstum)

2 Miniprojekte realisieren

Löst im Alleingang möglichst viele der untenstehenden Problemstellungen.

2.1 Projekt 01:

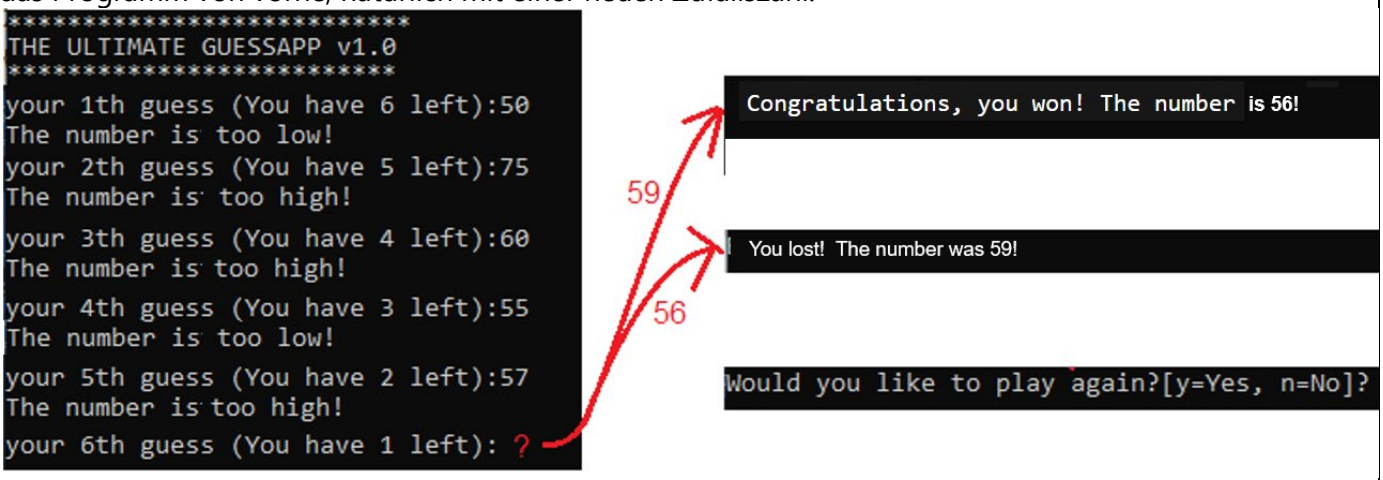
Projekt 01: Zahlen von bis ausgeben		
Schwierigkeitsgrad: ⭐	⏳ Zeit: 20 Minuten	Arbeitsform: Alleine 👤
Schreiben Sie ein Programm, das den Benutzer nach einer Start- und einer Endzahl fragt. Wenn die Endzahl kleiner oder gleichgross wie die Startzahl ist, soll eine Fehlermeldung ausgegeben und das Programm abgebrochen werden. Wenn die Startzahl und die Endzahl korrekt sind, sollen alle Zahlen von der Start- zur Endzahl untereinander ausgegeben werden		

2.2 Projekt 02:

Projekt 01: Ein Ratespiel

Schwierigkeitsgrad: ⭐ ⭐	Zeit: 40 Minuten	Arbeitsform: Alleine

Deine Aufgabe ist ein Ratespiel zu programmieren. Der Computer generiert eine zufällige ganze Zahl zwischen 1 und 100. Du hast sechs Rateversuche um die Zahl zu erraten. Nach jedem deiner Versuche erhältst du vom Programm ein Feedback, ob du richtig geraten hast oder nicht. Das Programm gibt auch einen Tipp, ob der Rateversuch zu hoch oder zu tief lag.
Wenn es dir innert dieser Anzahl Versuchen nicht gelingt, wird mitgeteilt, dass du verloren hast. Am Ende eines Spiels wirst du gefragt, ob du noch einmal Raten möchtest oder nicht. Wenn du Ja sagst, beginnt das Programm von vorne, natürlich mit einer neuen Zufallszahl.



Tipp 1:
Die Funktion randint(a, b) gibt eine ganze Zahl zwischen a und b (inklusive) zurück.

```
import random
zahl = random.randint(1, 100)
print(zahl)
```

Tipp 2: Pseudocode der helfen könnte, falls du Schwierigkeiten bei der Umsetzung hast:

```
//Geheimzahl erstellen und in secNo speichern

//Durchlaufe Schleife von i=1 bis i=5
//START Schleife
    //Benutzer nach Rateversuch i fragen
    //Benutzereingabe in guessNo speichern
    //Falls guessNo gleich secNo
        // Gratulation ausgeben
        //Schleife verlassen (break)
    //Falls guessNo > secNo
        // Rückmeldung "die Zahl ist zu hoch"
    //Andernfalls
        //Rückmeldung "die Zahl ist zu tief"
//ENDE SCHLEIFE
//Falls guessNo nicht gleich secNo
    //Ausgabe "Nicht geschafft" plus SecNo ausgeben
```

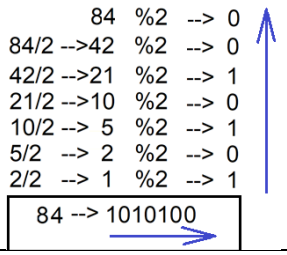
2.3 Projekt 03:

Projekt 03: Eine Dezimalzahl ins Binärsystem umrechnen

Schwierigkeitsgrad: ⭐ ⭐	Zeit: 40 Minuten	Arbeitsform: Alleine

Schreiben Sie ein Programm, das eine Dezimalzahl zwischen 0 und 99'999 entgegennimmt und diese Zahl im Binärsystem ausgibt.
Gibt man zum Beispiel die Dezimalzahl 328 ein, gibt die App als Resultat 101001000 aus.

```
Dezimalzahl von 0 bis 99'999: 328
Binaerzahl :101001000
```



Tipp:
Mann kann dieses Problem auf versch. Arten lösen. In meiner Lösung bin ich so vorgegangen, dass ich die Dezimalzahl (hier z.B. 84) immer zuerst Modulo 2 rechne (ergibt die Binärziffer) und danach mit der ganzzahligen Division durch 2 die Dezimalzahl mindestens halbiere und solange weiterfahre bis die Dezimalzahl 0 ist

Testfälle zur Überprüfung der Korrektheit:

Test-Nr	Eingabe (Dezimalzahl)	Erwartung
01	84	101 0100
02	125	111 1101
03	1'587	110 0011 0011
04	6'897	1 1010 1111 0001
05	9'856'781	1001 0110 0110 0111 0000 1101

Weitere Testdaten können Sie mit dem Windowscalulator (Ansicht Programmierer) erstellen.

2.4 Projekt 04:

Projekt 04: Primzahlen ausgeben

Schwierigkeitsgrad: ⭐ ⭐	Zeit: 30 Minuten	Arbeitsform: Alleine

Primzahlen sind natürliche Zahlen grösser als 1, die nur durch 1 und sich selbst teilbar sind, also keine anderen ganzzahligen Teiler besitzen.
Primzahlen bis 100:
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97

- 4 ist keine Primzahl, weil 4 = 2 × 2

9 ist keine Primzahl, weil 9 = 3 × 3

Schreib ein Programm, das den Benutzer zur Eingabe einer Zahl auffordert. Das Programm soll danach alle Primzahlen von 2 bis zur eingegebenen Zahl ausgeben.

2.5 Projekt 05:

Projekt 01: ggT und kgV von zwei Zahlen bestimmen

Schwierigkeitsgrad: ⭐⭐⭐	Zeit: 40 Minuten	Arbeitsform: Alleine

Bei der Bruchrechnung hast du den Einsatz des ggT (grösster gemeinsamer Teiler) und des kgV (kleinstes gemeinsames Vielfaches) gelernt.
Schreibe sie eine App, die den Benutzer auffordert zwei ganze Zahlen einzugeben. Berechne danach für die eingegebenen Zahlen den ggT und das kgV und gib diese Informationen dem Benutzer aus.

Der **ggT** ist die grösste Zahl, durch die sowohl A als auch B ohne Rest teilbar sind. Das **kgV** ist die kleinste Zahl, die sowohl ein Vielfaches der ersten Zahl als auch ein Vielfaches der zweiten Zahl ist.

Für die Berechnung des ggT sollen sie dabei den Algorithmus nach Euklid umsetzen. Dieser funktioniert wie folgt:
Man teilt die grössere durch die kleinere Zahl. Geht die Division auf, ist der Divisor der ggT. Geht die Division nicht auf, bleibt ein Rest. Dieser Rest ist der neue Divisor. Der alte Divisor wird zum Dividenden. Nun setzt man das Verfahren fort. Nach endlich vielen Schritten erhält man den ggT. Da das kleinste gemeinsame Vielfache (kgV) zweier Zahlen der Quotient aus ihrem Produkt
und ihrem ggT ist, kann man auf der Basis des ggT danach sehr einfach das kgV berechnen.

ggT von 544 und 391?

544 : 391 = 1; Rest 153
391 : 153 = 2; Rest 85
153 : 85 = 1; Rest 68
85 : 68 = 1; Rest 17
68 : 17 = 4; Rest 0

der ggT von 544 und 391 ist 17
Daraus folgt:
Das kgV von 544 und 391 ist
$(544 \cdot 391) : 17 = 12512.$

ggT von 13 und 7 ?

13 : 7 = 1; Rest 6
7 : 6 = 1; Rest 1
6 : 1 = 6; Rest 0

der ggT von 13 und 7 ist 1
Daraus folgt:
Das kgV von 13 und 7 ist
$7 \cdot 13 = 91.$

Testfälle zur Überprüfung der Korrektheit:

Test-Nr	Eingabe (Zahl 1)	Eingabe (Zahl 2)	Erwartung	
01	544	391	ggT= 17	kgV=12512
02	30	45	ggT= 15	kgV=90
03	12	150	ggT= 6	kgV=300
04	44	28	ggT= 4	kgV=308
05	145	30	ggT= 5	kgV=870