FH CAMPUS WIEN
UNIVERSITY OF APPLIED SCIENCES

DIE HOCHSCHULE, DIE MEHR KANN.

ILV Einführung in das Programmieren

Kontrollstrukturen

Mohamed Goha, BSc. | WS 2024/25

APPLIED LIFE SCIENCES | MASTERSTUDIENGANG BIOINFORMATIK

FH CAMPUS WIEN
UNIVERSITY OF APPLIED SCIENCES

Kapitelübersicht

- Motivation
- Bedingungen
- Verzweigungen
- Schleifen

Motivation Kontrollstrukturen

> Bisher: einfache Zuweisungen zu Variablen und Ein-/Ausgaben

> Keinerlei Logik, nur ein einziger **Programmablauf** möglich, egal wie der Input aussieht

> Kontrollstrukturen ermöglichen es, Programmablauf **abhängig von Eingaben und Umgebungsvariablen** zu steuern.

Motivation Kontrollstrukturen

- Wir möchten **Entscheidungen** treffen können, die von **Bedingungen** (conditions) abhängig sind. Diese Entscheidungen sollen den **Programmablauf** führen.
- Um das zu tun, können wir entweder
  - **Verzweigungen (branches)** oder
  - **loops (Schleifen)** verwenden.
- Python documentation zu Kontrollstrukturen: https://docs.python.org/3/tutorial/controlflow.html

Bedingungen

- Entscheidungen resultieren meist aus **Audrücken (expressions)**, denen ein boolean zugewiesen ist. (Jeder Ausdruck hat einen Wert)
  - ```
    print(10 < 5) # this prints "False"
    ```
- Diese Ausdrücke nennt man **boolean expressions**
- Oft sind boolean expressions Vergleiche
- https://docs.python.org/3/library/stdtypes.html#comparisons
    ```

Python Wahrheitswerte

> Der Wert “None”, 0 und leere Sequenzen (strings, etc) werden in Python standardmäßig als False interpretiert wenn sie als boolean expressions evaluiert werden:

```
empty_string = ""
if empty_string:
    print("this will not be executed!")
```

> Andere Werte standardmäßig True

If - Else Statements (Verzweigungen)

> Durch das if - statement können wir Bedingungen prüfen

> Beispiel: Achterbahnfahrt mit Mindestgröße 140cm

```
minimum_height = 140.0
height = float(input("Bitte geben Sie Ihre Körpergröße ein:"))

if height < minimum_height:
    print("Minimum für Körpergröße nicht erreicht. Sie können leider nicht an der Fahrt teilnehmen!")
else:
    print("Minimum für Körpergröße erreicht. Viel Spaß bei der Fahrt!")
```

> Ist der Wert der Bedingung “True”, so wird der Code ausgeführt, der im if-Block angeführt ist. Ansonsten, der Code im else - Block.

Exkurs Code Blöcke

> Es gibt verschiedene Arten, einen solchen Block einzugrenzen.

> Beispiel C: geschwungene Klammer

```c
if(a<b){
    // do something
}
```

> Python: Doppelpunkt deutet neuen Block an, Einrückung (**indentation**) grenzt ihn ein

> Meist 4 Leerzeichen

```python
a, b = 1,2

# new code block
if a < b :
    print("I'm inside the if - statement block!")

# code outside the code block
print("I'm outside the if - statement block!")
```

Else if / switch / match statement

> Erlaubt mehr Granularität für Bedingungen

```c
int day = 3;

switch(day) {
    case 1:
        printf("Monday");
        break;
    case 2:
        printf("Tuesday");
        break;
    case 3:
        printf("Wednesday");
        break;
    default:
        printf("Invalid day");
}
```

```python
# Python

x = 10
if x > 5:
    print("x is greater than 5")
elif x == 5:
    print("x is equal to 5")
else:
    print("x is less than 5")
```

```python
fruit = "apple"

# only python 3.10 or newer!
match fruit:
    case "apple":
        print("This is an apple")
    case "banana":
        print("This is a banana")
    case _:
        print("Unknown fruit")
```

> Code in default/else/”_” Block wird ausgeführt, falls keine der anderen explizit angegebenen Bedingungen True ist.

Exkurs Logik: Wann ist ein Ausdruck True/False?

P	Q	$P \wedge Q$	$P \vee Q$	$P \veebar Q$	$P \barwedge Q$	$P \Rightarrow Q$	$P \Leftarrow Q$	$P \Leftrightarrow Q$
T	T	T	T	F	T	T	T	T
T	F	F	T	T	F	F	T	F
F	T	F	T	T	F	T	F	F
F	F	F	F	F	T	T	T	T
P	Q	$P \wedge Q$	$P \vee Q$	$P \veebar Q$	$P \barwedge Q$	$P \Rightarrow Q$	$P \Leftarrow Q$	$P \Leftrightarrow Q$
		AND (conjunction)	OR (disjunction)	XOR (exclusive or)	XNOR (exclusive nor)	conditional "if-then"	conditional "if"	biconditional "if-and-only-if"

where T *means* **true** *and* F *means* **false**

Operatoren

```
if a < b and b <= c:
    print("Both conditions are True")

if x is None or y == "Hello":
    print("One or both conditions are True")

if not is_active:
    print("The user is not active")

if x is y and x != 20:
    print("x is y and x is not 20")

if not a or b == 1:
    print("At least one condition is True")

if obj1 is not obj2 and len(obj1) == len(obj2):
    print("The objects are not the same but have equal length")
```

Operation	Result	Notes
`x or y`	if *x* is true, then *x*, else *y*	(1)
`x and y`	if *x* is false, then *x*, else *y*	(2)
`not x`	if *x* is false, then `True`, else `False`	(3)

Operation	Meaning
`<`	strictly less than
`<=`	less than or equal
`>`	strictly greater than
`>=`	greater than or equal
`==`	equal
`!=`	not equal
`is`	object identity
`is not`	negated object identity

Operatoren

- **or:** Falls erste boolean expression schon True ist, wird zweite nicht mehr geprüft
- **and:** Falls erste boolean expression schon False ist, wird zweite nicht mehr geprüft.
- Reihenfolge kann wichtig sein - Beispiel Division durch 0:

```
a = 0
b = 10

b / a > 100 or a == 0 # results in error
a == 0 or b / a > 100 # no error
```

Schleifen (loops)

> In manchen Fällen sollen Teile des Codes wiederholt werden
> Wie oft / wie lange das passieren soll, **entscheiden** wir wieder durch eine oder mehrere **Bedingung(en)**
> Schleifen erlauben es uns, den gleichen code mehrmals auszuführen
> Code muss dadurch nur einmal geschrieben werden, weniger **code duplication**
> Können je nach Bedingung gar nicht, mehrmals, oder unendlich oft ausgeführt werden

While Schleife

- Wiederholt die Ausführung eines Code - Blocks solange eine Bedingung True ist
- **Achtung**: Wenn der Wahrheitswert der Bedingung durch den Code - Block nie zu False geändert wird, kann es zu einer **Endlosschleife** kommen - das Programm terminiert nicht von selbst
  - Manchmal ist das wünschenswert, z.B. Server

While Schleife

> Beispiel:

```
number = -1

while number <= 0:
    number = float(input("Please enter a positive number: "))
    if number < 0:
        print("That's a negative number or zero. Try again!")
    elif number == 0:
        print("You entered zero. Try again!")

print(f"Thank you! You entered: {number}")
```

For Schleife

- In Python: wiederholt einen Code-Block für jedes Element in einem **iterable** an elementen (strings, Datenstrukturen, etc.)
- Beispiel:
  - Für jeden Buchstaben im DNA Strang `sample_dna`, berechne das dazugehörige Nukleotid im komplementären Strang

For Schleife: Beispiel

> Wir **iterieren** über variable some_string
> Bei der ersten Iteration ist der Buchstabe “I” der Variable “char” zugewiesen, bei der zweiten ein Leerzeichen, usw.
> Wir machen char zu einem Kleinbuchstaben und fügen ihn zu einem neuen string “lowercase” hinzu. Diesen geben wir nach der Schleife aus.

```
some_string = "I am a string!"
lowercase = ""

# convert all characters in string to lowercase
for char in some_string:
    lowercase += char.lower()
print(lowercase)
```

Manuelle Schleifenkontrolle

- **Iterationsabbruch**
  - Ausdruck `continue` führt direkt zur loop condition zurück, und führt die nächste Iteration aus, falls condition true ist
  - Code nach Ausdruck (innerhalb der Schleife) wird für diese Iteration nicht ausgeführt
- **Schleifenabbruch**
  - Ausdruck `break` bricht Schleife ab
  - Code nach Ausdruck (innerhalb der Schleife) wird nicht ausgeführt, Sprung zum Code, der nach der Schleife ausgeführt werden soll

Verschiedene while und for Schleifen

- **Zählschleife:**
  - Zähle von <Anfangszahl> bis <Endzahl> mit Schrittweite <Schrittweite>, und führe jedesmal den Anweisungsblock A aus.
  - Der aktuelle Zählerwert ist über die Variable i verfügbar.

```
for i := <Anfangszahl> to <Endzahl> stop <Schrittweite> do
   <Anweisungsblock A>
end for
```

- **Foreach-Schleife (Mengenschleife):**
  - Führe den Anweisungsblock A für jedes Element element der Menge Menge aus.
  - Das aktuelle Element ist über die Variable element verfügbar.
  - Python for Schleife ist im Prinzip eine foreach-Schleife

```
for each element in Menge do
   <Anweisungsblock A>
end for
```

Verschiedene while und for Schleifen

- **Kopfgesteuerte Schleife:**
  - Bedingung wird vor der Iteration ausgewertet.
  - Es kann passieren, dass der Schleifenkörper nie ausgeführt wird.

```
while <Bedingung B> do
   <Anweisungsblock A>
end while
```

- **Fußgesteuerte Schleife:**
  - Bedingung wird nach der Iteration ausgewertet.
  - Der Schleifenkörper wird mindestens einmal ausgeführt.

```
repeat
   <Anweisungsblock A>
until <Bedingung B> {Schleifenfuß}
```