1. Einführung in die Programmiersprache "Python"

Voraussetzungen:

- Grundbegriffe (Informatik, Informatiksystem, Teilgebiete, EVA, Algorithmus)
- Codierung Text (ASCII), natürliche Zahlen (dezimal, dual, hexadezimal, Positionssystem)
- Codierung Sound (grob), Bilder (pixel- und vektororientiert, grob)
- Erstellung von Algorithmen mittels Flussdiagramm/Struktogramm
- Anweisungen, Variablenbegriff (Name, Typ, Wert)
- Kontrollstrukturen (Sequenz, Selektion, kopf-/fußgesteuerte Iteration)
- Zerlegungsprinzip (Top-Down-Methode), Prozeduren, Funktionen
- Rekursion, Schreibtischtest (bevorzugt Tabellenform)

Download-Quellen:

https://www.python.org/
http://winpython.github.io/#portable

Installationsversion:

- Datei **python-versionsnummer.exe** ausführen, Häkchen bei "Path" und "All Users"
- installiert eine Kommandozeilen-Umgebung (shell), eine einfache Entwicklungsumgebung (IDLE) und einige zusätzliche Befehls-Bibliotheken
- registriert Dateien mit der Python-Erweiterung (.py) unter Windows, so dass diese einfach durch Doppelklick gestartet werden bzw. über das Kontextmenü im Editor (IDLE) geöffnet werden können

Portable Version:

- Ordner **PythonPortable** auf Datenträger speichern (USB-Stick, Festplatte o.ä.)
- enthält eine einfache Entwicklungsumgebung (IDLE) und einige zusätzliche Tools und Befehls-Bibliotheken
- Python-Dateien (.py) muss man im zuerst gestarteten Editor öffnen, um sie zu editieren oder auszuführen, da keine Windows-Registrierungen erzeugt werden
- Start des Editors über die Datei: **IDLEX (Python GUI).exe** im Ordner **PythonPortable**

Eine Entwicklungsumgebung mit Quelltexteditor, Fehlersuchtool (Debugger), Ausführungsmöglichkeit, Projektverwaltung u.ä. nennt man IDE (Integrated Development Environment). IDLE ist eine sehr einfache IDE. Komplexere IDE (z.T. kostenpflichtig) sind z.B. Eric, PyCharm, Eclipse, WingIDE.

Nutzung:

shell: Befehle eingeben (z.B. `x=5, print(x), y=input(), print(y), print("ABC"), 23+17, 2.45+3.65` usw.), Aufruf der Hilfe über `help()`, Verlassen der Hilfe mit `quit`

idle: - zeigt zunächst eine ***shell*** an (mit Menüleiste, farbiger Syntax - Syntaxhighlightning), Befehlseingabe wie in der normalen ***shell***
- über **File >>> New File** Aufruf des ***Quelltexteditors***
- dort Eingabe kompletter Programme, Start über **Run >>> Run Module (F5)**, Ausführung in der ***shell*** (bzw. bei Fensterprogrammen in eigenem Fenster)

cmd: - Kommandozeilen-Umgebung von Windows, **Start >>> Ausführen >>>** `cmd` (bzw. "Eingabeaufforderung")
- Wechsel in das jeweilige Verzeichnis mit dem Python-Quelltext (`LW:, cd\, cd.., cd verzeichnis, dir`), dann `dateiname.py` und `Enter` (oder `python dateiname.py` und `Enter`) - startet das jeweilige Python-Programm (**Voraussetzung:** Python ist in den Windows-Pfad eingetragen, nicht bei portabler Nutzung)
- nur `python` startet eine Python-Shell, Verlassen der Shell und Rückkehr zu Windows mit `quit()` oder `STRG+Z`

Beispiele zeigen und ausprobieren:

Beispielprogramme zeigen (aus idle oder von Konsole oder mit Doppelklick starten)
Befehlseingaben aus shell, idle, cmd zeigen (interaktiv)

2. Was für eine Sprache ist "Python"?

Geschichte:

Die Sprache wurde Anfang der 1990er Jahre von **Guido van Rossum** am Zentrum für Mathematik in Amsterdam entwickelt. Der Name der Programmiersprache Python hat nichts mit Schlangen zu tun. Für Guido van Rossum stand vielmehr die britische Komikertruppe Monty Python mit ihrem legendären Flying Circus Pate für den Namen.

Guido van Rossum schrieb 1996 über die Entstehung des Names seiner Programmiersprache:

"Vor über sechs Jahren, im Dezember 1989, suchte ich nach einem 'Hobby'-Programmier-Projekt, dass mich über die Woche um Weihnachten beschäftigen konnte. Mein Büro ... war zwar geschlossen, aber ich hatte einen PC und sonst nichts vor. Ich entschloss mich einen Interpreter für die neue Skripting-Sprache zu schreiben, über die ich in der letzten Zeit nachgedacht hatte: ein Abkömmling von ABC, der UNIX/C-Hackern gefallen würde. Python hatte ich als Arbeitstitel für das Projekt gewählt, weil ich in einer leicht respektlosen Stimmung war (und ein großer Fan von Monty Python's Flying Circus)."

Was für eine Sprache ist Python? Python ist:

- keine Skriptsprache, aber man kann damit **Skripte** schreiben (Skripte bezeichnen Programme, mit denen man andere Programme aufruft, Betriebssystem-Aufrufe macht, Geräte anspricht, Dateien im Dateisystem manipuliert . . .)
- eine **objektorientierte** Sprache, aber man muss keine Objekte (explizit) benutzen
- keine Sprache, mit der man Programme mit **grafischen Oberflächen** schreibt, aber man kann das trotzdem sehr gut, weil es viele grafische Bibliotheken gibt
- eine der drei großen P-Sprachen: perl, php und eben Python. Python ist wirklich eine Sprache, mit der man **dynamische Web-Seiten** erstellen kann. Und viele machen das.
- **eine der am häufigsten benutzten Sprachen** im WWW. Wenn man sich fragt, in welcher Sprache die Programme geschrieben sind, die weltweit am häufigsten benutzt werden, dann kommt man auf Python. Zwei Unternehmen, die Python wesentlich benutzen, sind **Google** und **YouTube**.

Was sind also die Vorteile von Python als Programmiersprache?
(Verwendungs- und Beliebtheitsindizes und Programmbeispiele in anderen Sprachen zeigen!)

- Python-Code ist **gut lesbar**. Damit sind Python-Programme gut **nachvollziehbar**, damit gut zu verändern und zu verbessern (Wartbarkeit).
- Python-Code unterstützt die **Wiederverwertbarkeit** von Code durch Objektorientiertheit und die **Aufteilung von Code** in Module.
- Python-Code ist **plattformunabhängig**. Ein auf einem Rechner geschriebenes Programm ist meistens ohne Veränderung auf einem anderen Rechner lauffähig.
- Python bringt schon eine große Zahl von **Bibliotheken** mit, viele weitere können aus dem Internet besorgt werden.
- Python-Code ist im Verhältnis zu Java-Code **relativ kurz**, im Verhältnis zu perl-Code **sehr klar**. Als Folge davon ist Python schnell bei der Entwicklung und Erstellung von Programmen. Man muss oft keine Klimmzüge machen (wie in C oder Java), um einfache Probleme zu lösen.
- Python ist **freie Software**. Es ist in den meisten Linux-Distributionen schon enthalten, zu Windows frei aus dem Internet herunterzuladen.
- Python-Programme können wie ganz normale Programme **ausgeführt** werden, es gibt aber auch Test-/Entwicklungsumgebungen, in der man Programm-Fragmente leicht und ohne großen Aufwand testen kann. Python-Code kann man interaktiv auf der Shell/Kommandozeile schreiben.
- Python hat eine gute **Datenbank-Schnittstelle.**
- **Die meisten Python-Distributionen haben schon eine grafische Bibliothek** dabei, nämlich Tkinter. Die hat den Vorteil, dass sie freie Software ist.

3. Erste Schritte im interaktiven Modus (shell, idle) -

Literale, Datentypen, Variablen, Ausdrücke, Built-In-Funktionen, Eingaben, Ausgaben

Bemerkungen:

- in der Regel nur eine Anweisung pro Zeile - !!! bessere Übersichtlichkeit !!!:

```
print("Hallo")
```

- falls nötig, mehrere Anweisungen pro Zeile, dann durch Semikolon trennen:

```
x=5 ; print(x)
```

(Die Anweisungen entsprechen dann in Programmen der gleichen Einrücktiefe!)

- falls nötig (z.B. lange Ausgabetexte), Trennung der Befehle durch Backslash \ möglich

```
print("Das ist ein langer Befehl, der vielleicht " \
      "nicht in eine Zeile passt und daher getrennt " \
      "werden sollte!")
```

- Setzt man den Cursor in eine bereits abgearbeitete Zeile von **idle** und drückt **Enter**, so wird diese Zeile kopiert und als neue Anweisung eingefügt, die man dann vor der Ausführung noch editieren kann.

Werte (Instanzen) von Datentypen (ganze Zahlen, Zeichenketten, Kommazahlen usw.) müssen in bestimmter Weise geschrieben werden, um als solche erkannt zu werden. Diese Schreibweisen nennt man ***Literale***.

Ganze Zahlen:

```
-9          +234        2054
(12-17)     3*(5-7)     24/6
18/4  >>> liefert eine Gleitkommazahl (4.5)
18//4 >>> liefert eine ganze Zahl (4)
```

Gleitkommazahlen:

```
23.78       -0.0045     +3.5
2.5-0.09    (34.8+45)/3.2
```

Zeichenketten (Strings):

```
"Hallo Welt!"           'Hallo Welt!'
"Hallo"+" "+"Welt"+"!"  >>> liefert 'Hallo Welt!
```

Listen - geordnete Ansammlungen von Elementen beliebiger Datentypen:

```
[1,2,3,4,5]       ["Hallo",4.5,3648,"Paul","x"]
["A","B","C"]+[1,2,3,4] >>> liefert ['A', 'B', 'C', 1, 2, 3, 4]
```

Dictionarys - Ansammlung von Zuordnungspaaren - Schlüssel und zugehöriger Wert:

```
{"Name": "Paul", "Alter": 18, "Groesse": 1.78}
{1: "Paul", 2: 18, 3: 1.78, "Wohnort": "Finsterwalde", 5.5: "Hallo"}
```

Variablen sind ***Platzhalter*** für Werte eines bestimmten ***Datentyps***. Sie werden oft auch ***Bezeichner*** genannt. Mit der Zuordnung des Datentyps werden sowohl die möglichen Werte der Variable, als auch die zulässigen Operationen festgelegt. Python ist jedoch ***dynamisch typisiert***, d.h. der Typ einer Variable kann sich im Laufe des Programms ändern.

Beispiel:

```
>>> x=5 ; type(x)
<class 'int'>
>>> x="Hallo" ; type(x)
<class 'str'>
>>> x=input()
23.4
>>> type(x)
<class 'str'>
>>> x=float(x) ; type(x)
<class 'float'>
>>>
```

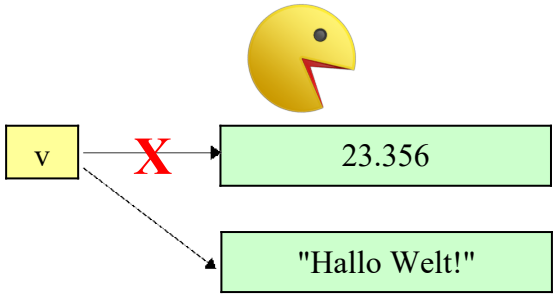
Mit der Funktion `type(variable)` kann man den Datentyp einer Variable feststellen.

Folgerungen der dynamischen Typisierung:

- kein Deklarationen mit Datentypangabe notwendig (wie in Java, C, Delphi etc.)
- Typfestlegung geschieht bei der ***Zuweisung*** des Wertes (einfacheres Programmieren)
- Typprüfungen meistens erst zu Programmlaufzeit (Fehlererkennung schwieriger)

Ursachen der dynamischen Typisierung:

Variablen verweisen in Python stets auf Objekte (Werte, Wertesammlungen), d.h. nicht die Variablen haben einen Datentyp, sondern nur die referenzierten Objekte. Durch Zuweisung kann eine Variable ein anderes Objekt referenzieren und damit auf einen Wert eines anderen Datentyps verweisen.




```
v=23.356
# v zeigt auf das Objekt mit dem Wert 23.356
# vom Typ float
v="Hallo Welt!"
# v zeigt jetzt auf das Objekt mit dem Wert
# "Hallo Welt!" vom Typ str, der sogenannte
# GarbageCollector gibt den Speicher des ersten,
# nicht mehr referenzierten Objekts wieder frei,
# löscht dieses also (falls keine andere Variable
#  mehr darauf zeigt)
```

Zuweisung: `variablenname=wert` oder `variablenname=ausdruck`

Ausdrücke: sind arithmetische Ausdrücke, logische Ausdrücke, Ergebnisse von Funktionsaufrufen, zusammengesetzte Ausdrücke (enthalten Werte, Operatoren, Funktionen, Klammerungen u.ä.)

Beispiele:

```
x=23.45+0.76
y="Hallo " + "Welt!"
zahl=int("4565")
maximum=max([2,6,9,12,43,-6,9])
erg=input()
erg=(4<5) or (5>7)
```

`print()`, `max()`, `input()` usw. sind vorgefertigte Funktionen (***Built-In-Functions***).

Zulässige Bezeichnernamen *(Variablen, Funktionen, Klassen, ...)*:

- Großbuchstaben, Kleinbuchstaben
- Ziffern (nicht an erster Stelle)
- Unterstrich (_ , auch an erster Stelle), kein Leerzeichen
- Groß- und Kleinschreibung wird unterschieden (case-sensitive).
- Umlaute sind erlaubt, sollten aber vermieden werden.

```
wurzel      Berechnung_Mittelwert        _ABC_125        fktSumme
```

Wichtige Operatoren:

arithmetische	relationale	logische
`+ , - , * , /`	`== , !=` (gleich, ungleich)	`and , or , not`
`+ , -` (auch als Vorzeichen)	`< , <= , > , >=`	
`//` (Ganzzahldivision, `8//5=1`)	`in` (Element von)	
`%` (Divisionsrest, `8%5=3`)		`not (3<4)`
`**` (hoch, `2**3=8`)	`4 in [2,4,1]`	`(3<4) or (4<2)`

Die Rangfolge der Operatoren (Bindungspriorität) ist festgelegt. Auf Grund der besseren Lesbarkeit und der möglichen Unkenntnis der Rangfolge sollten Klammern verwendet werden!

Ausgaben: `print()` - Funktion

```
>>> print(2,5,3.87,"Hallo")
2 5 3.87 Hallo
>>> print(2,5,3.87,"Hallo") ; print("Guten Tag")
2 5 3.87 Hallo
Guten Tag
>>> print(2,5,3.87,"Hallo", end=" ") ; print("Guten Tag")
2 5 3.87 Hallo Guten Tag
>>> print(2,5,3,sep="")
253
>>> print(2,5,3,sep=".",end=" ") ; print("Hallo")
2.5.3 Hallo
```

- Ausgabebestandteile trennt man durch Komma.

- `print()` endet normalerweise mit einem Zeilenumbruch.
  Diesen kann man mit `end="..."` durch `...` ersetzen.

- `print()` schreibt zwischen Werte normalerweise ein Leerzeichen.
  Dieses kann man mit `sep="..."` durch `...` ersetzen.

Eingaben: `input()` - Funktion

```
>>> x=input()
23.4
>>> type(x)
<class 'str'>
>>> y=float(x)
>>> type(y)
<class 'float'>
>>> print("Eingabe: ",end="") ; x=input()
Eingabe: 234
>>> x=input("Eingabe: ")
Eingabe: 234
>>> x=int(input("Ganze Zahl: ")) ; type(x)
Ganze Zahl: 45
<class 'int'>
>>> x=eval(input("Liste eingeben: "))
Liste eingeben: ["Hallo",23,3.47,"ABC"]
>>> print(x)
['Hallo', 23, 3.47, 'ABC']
>>> x=eval(input("Dictionary eingeben: "))
Dictionary eingeben: {"Name": "Paul", "Alter": 23, "Größe": 1.78}
>>> print(x)
{'Alter': 23, 'Größe': 1.78, 'Name': 'Paul'}
```

- `input()` - Eingaben schließt man mit Enter ab.
- Der Eingabetyp ist immer `str` (String).
- Der `input()` - Funktion kann man als Argument einen vorher auszugebenden String übergeben (Eingabeaufforderung).
- Eingaben kann man casten und damit eine Typumwandlung vornehmen (***Casting***).

`y=int(y)`	- wandelt y in einen Integerwert um
`y=float(y)`	- wandelt y in eine Gleitkommazahl um
`y=str(y)`	- wandelt y in einen String um
`y=eval(y)`	- untersucht (evaluiert) y und wandelt es in den entdeckten Typ um (int, float, liste, dictionary)

- Das Casting kann man gleich als Funktion auf `input()` anwenden.

```
y=int(input("Ganze Zahl eingeben: "))
```

- Mit der `eval()` - Funktion evaluiert man eine Eingabe, d.h. ein möglicher Datentyp (außer String) wird erkannt und gecastet (auch komplexerer Datentypen wie Listen, Dictionarys, Tupel - *ähnlich wie Listen, aber runde Klammern, weniger Funktionalität*).

```
>>> y=eval(input("Etwas eingeben: "))
Etwas eingeben: "Hallo",2.34,5674,"Paul"
>>> type(y)
<class 'tuple'>
>>> print(y)
('Hallo', 2.34, 5674, 'Paul')
```

4. Erste einfache und strukturierte Programme (idle und editor)

Einfache sequentielle Programme benötigen keine äußere Rahmenstruktur und können im Editor einfach zeilenweise geschrieben werden.
Programme sollten ***kommentiert*** werden (*Kopf mit Autor, Datum, Programmbeschreibung / wichtige Zeilen mit kurzer Erklärung / ggf. Trennlinien u.ä.*).

```
# Das ist ein Kommentar, der bis zum Zeilenende verläuft.

""" Das ist ein Kommentar, der umfangreicher sein kann und
über mehrere Zeilen gehen kann. """
```

Aufgabe: Programmiere einen kleinen Taschenrechner, der für zwei einzugebende ganze Zahlen alle möglichen Grundoperationen ausführt und die Ergebnisse ausgibt!

```
# Autor: C. Lehmann
# Datum: 08.01.2017
# Zweck: Ein erstes sequentielles Programm.

print("Mein kleiner Taschenrechner")
print("---------------------------")
print()
print("Gib zwei ganze Zahlen ein!")
print()
a=int(input("a = "))            # Eingabe und Casting
b=int(input("a = "))
print()
summe=a+b                       # Berechnungen
differenz=a-b
produkt=a*b
quotient=a/b
gquotient=a//b
grest=a%b
potenz=a**b
print("Summe =",summe)          # Ausgaben
print("Differenz =",differenz)
print("Produkt =",produkt)
print("Quotient =",quotient)
print("Ganzzahliger Quotient =",gquotient)
print("Ganzzahliger Rest =",grest)
print("Potenz =",potenz)
```

Aufgabe: Schreibe ein Programm, das 3 Zahlen nacheinander in eine Liste einliest, die Liste ausgibt und die Summe der Zahlen berechnet und ausgibt. Der Zugriff auf ein Listenelement erfolgt über `listenname[index]`, wobei der Index bei 0 beginnt.

```
# Autor: C. Lehmann
# Datum: 08.01.2017
# Zweck: 3 Zahlen in Liste einlesen und Summe berechnen.

liste=[]
zahl=float(input("1. Zahl = "))
liste=liste+[zahl]
zahl=float(input("2. Zahl = "))
liste=liste+[zahl]
zahl=float(input("3. Zahl = "))
liste=liste+[zahl]
print()
print("Die Liste ist:",liste)
print()
summe=liste[0]+liste[1]+liste[2]
print("Die Summe ist:", summe)
```

Kontrollstrukturen: steuern die Reihenfolge der Abarbeitung der Anweisungen

- Sequenzen (Abfolgen, siehe oben)
- Iterationen (Wiederholungen, Schleifen)
- Selektionen (Auswahlen, Verzweigungen)

Iterationen und Selektionen bestehen in der Regel aus einem (oder mehreren) ***Anweisungsköpfen*** (nicht eingerückt) und einem (oder mehreren) ***Anweisungskörpern*** (eingerückt).
Die Einrückungen geben die Blockstrukturen an, die in anderen Sprachen durch Klammerungen erreicht werden.
Kontrollstrukturen können ***geschachtelt*** werden, d.h. in einer Kontrollstruktur kann eine (oder mehrere) andere (vollständig) enthalten sein.

Einfache Varianten in Python:

`while Bedingung :`
`  Anweisung`
`  Anweisung`
`  .........`
	Solange die Bedingung erfüllt ist, führe die Anweisungen aus. Nach jedem Durchlauf der Anweisungen wird die Bedingung erneut geprüft.

`if Bedingung :`
`  Anweisung`
`  Anweisung`
`  .........`
	Wenn die Bedingung erfüllt ist, dann führe die Anweisungen genau einmal aus.

`if Bedingung :`
`  Anweisung`
`  Anweisung`
`  .........`
`else :`
`  Anweisung`
`  Anweisung`
`  .........`
	Wenn die Bedingung erfüllt ist, dann führe die Anweisungen genau einmal aus, sonst führe die anderen Anweisungen genau einmal aus.

Aufgabe: Setze das Ratespiel "1 aus 100" in Python um! Eine Zufallszahl `z` kann man mit `z=random.randint(1,100)` erzeugen. Dazu muss mit `import random` die Bibliothek `random` am Anfang des Quelltextes eingebunden werden.

```
import random

print("Ratespiel - 1 aus 100")
print("----------------------")
print()
z=random.randint(1,100)
print("Zufallszahl - 1 aus 100 - erzeugt!")
print()
i=0
r=0
while r!=z :
    r=input("Eingabe: ")
    r=int(r)
    i=i+1
    if r<z :
        print("Zahl zu klein!")
    if r>z :
        print("Zahl zu groß!")
print()
print("Richtig nach",i,"Versuchen!")
```