Input

- ```
  input(Nachricht) -> str
      druckt die Nachricht auf die Kommandozeile, nimmt dann beliebigen
      Text an bis der Nutzer Enter eingibt. Diesen Text returned er dann.

  >>> t = input("Text please: ")    # Leerzeichen beachten!
  Text please: Dieser Teil wird vom Nutzer eingegeben
  >>> t
  'Dieser Teil wird vom Nutzer eingegeben'
  ```

- ```
  open(Dateipfad, Leseform) -> file
      öffnet eine Datei die dann gelesen und beschrieben werden kann.
      Diese Datei muss im nachhinein wieder mit close() geschlossen werden.
      Leseformen sind:
          - 'r' : 'read', Datei kann nur gelesen werden (default)
          - 'w' : 'write', Datei kann nur beschrieben werden
          - 'a' : 'append', wie 'w' aber springt automatisch zum Ende
                  der Datei (falls sie existiert)
          - 'r+': 'read+', wie 'r', aber kann auch beschrieben werden
          - 'w+': 'write+', wie 'w', aber kann auch gelesen werden
          - 'a+': 'append+', wie 'a', aber kann auch gelesen werden
          - 'x' : kreiert die Datei, sonst nichts.
      'w+' und 'a+' unterscheiden sich von 'r+' in dem sie eine neue
      Datei herstellen falls keine unter dem Pfad vorhanden ist.

  >>> f = open("file.txt", 'r')
  >>> Zeile = f.readline()    # liest eine Zeile, setzt Lesepunkt am
                              # Ende diezer Zeile
  >>> Rest = f.read()         # liest die ganze Datei, setzt Lesepunk
                              # am Ende
  >>> Zeile
  'Inhalt der ersten Zeile\n' # readline beinhaltet auch das Symbol für
                              # eine neue Zeile, falls vorhanden
  >>> f.close()
  ```

Output

- ```
  print(Nachricht)
      druckt die Nachricht auf die Kommandozeile

  print(N1, N2, N3)
      druckt eine Nachricht nach der anderen auf die Kommandozeile

  print(N1, N2, N3, sep=)
      druckt eine Nachricht nach der anderen auf die Kommandozeile,
      verwendet sep als das/die Zeichen dazwischen

  print(N1, N2, N3, end=)
      druckt eine Nachricht nach der anderen auf die Kommandozeile,
      druckt am ende end

  >>> print("message")
  message
  >>> print("embedded", 2, "integer")
  embedded 2 integer
  >>> print("embedded", 2, "integer", sep=" : ")
  embedded : 2 : integer
  >>> print("embedded", 2, "integer", sep=" : ", end="NEWLINE")
  embedded : 2 : integerNEWLINE>>> print("ups")
  ups
  ```

- ```
  info(Nachricht)    # braucht libs.log
      macht eine Log entry mit Nachricht
      mögliche Funktionsnamen sind
          trace, debug, info, success, warning, error, critical
      default Formatting beinhaltet
      Zeit vergangen | log level | datei:function (Zeile) - Nachricht

  >>> info("message")
  0:00:00.011999 | INFO     |  example_logging:log (  15) - message
  ```

- ```
  f.write(Nachricht) -> int
      schreibt die Nachricht in die Datei f, gibt die Anzahl an
      geschriebenen Zeichen zurück

  >>> f = open("file.txt", 'w')
  >>> f.write("message")    # überschreibt falls schon Text vorhanden!
                            # verwende 'a' um ans Ende zu springen
  >>> f.close()
  ```
  ```

Modify

- ```
  ## Arithmetics
  >>> 3 + 2
  5
  >>> 3 - 2
  1
  >>> 3 * 2
  6
  >>> 3 / 2
  1.5
  >>> 3 // 2
  1
  >>> 3 % 2    # modulo, Rest von einer Division
  1
  >>> 3**2     # power, 3 hoch 2
  9
  >>> a = 5
  >>> a += 2   # alle auch in dieser Form equivalent zu
  >>> a = a + 2
  ```

- ```
  ## Strings
  >>> w = "world"
  >>> len(w)
  5
  >>> w[0]     # Element 0, das Zeichen an Stelle 0 des Strings
  'w'
  >>> "hello " + w    # Zusammensetzung von zwei Strings
  'hello world'
  >>> f"hello {w}"    # formatierte String
  'hello world'
  >>> f"hello {w.capitalize()}"    # {} dürfen beliebigen Code beinhalten
  'hello World'
  ```

Modify (cont.)

- ```
  ## Lists
  >>> numbers = list(ranger(10))
  >>> numbers
  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
  >>> numbers[-1]    # das erste Element von hinten gezählt
  9
  >>> sum(numbers)
  45
  >>> numbers[2] = 20
  >>> numbers
  [0, 1, 20, 3, 4, 5, 6, 7, 8, 9]
  >>> numbers[2:]    # sub-Liste, von Element 2 bis zum Ende
  [20, 3, 4, 5, 6, 7, 8, 9]
  >>> numbers[-3:]   # geht auch von Hinten
  [7, 8, 9]
  >>> numbers.pop()  # entfernt leztes Element und gibt es zurück
  9
  >>> numbers
  [0, 1, 20, 3, 4, 5, 6, 7, 8]
  ```

- ```
  ## Dictionaries
  >>> d = {"first": "1st", "second": "2nd"}
  >>> d["first"]
  '1st'
  >>> d["third"] = "3rd"
  >>> d2 = dict(enumerate(("1st", "2nd", "3rd")))
  >>> n = 20
  >>> print(f"{n} is the {d2[numbers.index(n)]} element in {numbers}")
  20 is the 3rd element in [0, 1, 20, 3, 4, 5, 6, 7, 8]
  ```

Conditionals

- ```
  if condition:
      Führt den danach eingerückten code aus falls 'condition' nicht
      als False ausgewertet wird. False sind:
          False
          None
          null (0 oder 0.0)
          leere iterierbare ("", (), [], {})
          falsche gleichheiten (==, <, >, <=, >=)

  >>> if True:
          # wird ausgeführt
  >>> if "" or []:
          # wird nicht ausgeführt
  >>> n = 3
  >>> if 1 < n < 5:
          # wird ausgeführt
  ```

- ```
  else:
      Führt den danach eingerückten code aus falls das if davor nicht
      als True ausgewertet wurde. True ist alles was nicht False ist.

  >>> if False:
          # wird nicht ausgeführt
      else:
          # wird ausgeführt
  ```

- ```
  elif condition:
      Führt den danach eingerückten code aus falls das if davor nicht
      als True ausgewertet wurde und 'condition' nicht False ist.

  >>> n = 3
  >>> if n == 4:
          # wird nicht ausgeführt
      elif n == 3:
          # wird ausgeführt
      elif n == 3:
          # wird nicht ausgeführt, da das Vorige if/elif schon war
      else:
          # wird nicht ausgeführt
  ```
  ```

Conditionals (cont.)

- match/case
  findet den ersten passenden Fall und verwendet ihn

```
>>> match (4, 0):
        case (0, 0):
            print("Ursprung")
        case (0, y):
            print(f"X-Achse, Y={y}")
        case (x, 0):  # der in diesem Fall passende case
            print(f"Y-Achse, X={x}")
        case (x, y) if x == y:
            print(f"X = Y = {x}")
        case (x, y):  # wird übersprungen wegen vorigem case
            print(f"X={x}, Y={y}")
        case _:       # falls nichts gematched wurde
            print("Kein 2D-Vektor")
```

- while condition:
  Führt den danach eingerückten code so lange 'condition' nicht als False ausgewertet wird.

```
>>> while True:
        # Endlosschleife
>>> n = 0
>>> while n < 3:
        n += 1
        print(n)
1
2
3
```

- for element in iterable:
  Führt den danach eingerückten code einmal für jedes Element in einer Iterierbaren aus.

```
>>> for n in [1, 2, 4]:
        print(n)
1
2
4
```