*Spicker gpr (2 A4 Seiten)*

|  |  |  |
| --- | --- | --- |
| Name | Type | Assignment |
| list | Sequence Type | x = ["apple", "banana", "cherry"] |
| \* list of list | Sequence Type | x = [[1,0,3],[0,2,4]] |
| set | Set Type | x = {"apple", "banana", "cherry"} |
| dictionary | Mapping Type | x = {"name" : "John", "age" : 36} |
| tuple | Sequence Type | x = ("apple", "banana", "cherry") |
| str | Text Type | x = "Hello World" |
| int | Numeric Type | x = 20 |
| float | Numeric Type | x = 20.5 |
| complex | Numeric Type | x = 1j |
| bool | Boolean Type | x = True |

\* kein eigener Typ

**Files lesen**

file = open(filename)

file.readline() #ist ein Iterator

**Exceptions**

*An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions. In general, when a Python script encounters a situation that it cannot cope with, it raises an exception. An exception is a Python object that represents an error.*

*When a Python script raises an exception, it must either handle the exception immediately otherwise it terminates and quits.*

Raising an exception

raise Exception('Fehlermeldung')

Handling Exceptions with try and except:

try:

You do your operations here;

except *ExceptionI*:

If there is ExceptionI, then execute this block.

except *ExceptionII*:

If there is ExceptionII, then execute this block.

else:

If there is no exception then execute this block.

* Generic exception can handle any exceptions

|  |  |
| --- | --- |
| **Exception** | Base class for all exceptions |
| **StopIteration** | Raised when the next() method of an iterator does not point to any object. |
| **SystemExit** | Raised by the sys.exit() function. |
| **StandardError** | Base class for all built-in exceptions except StopIteration and SystemExit. |
| **ArithmeticError** | Base class for all errors that occur for numeric calculation. |
| **OverflowError** | Raised when a calculation exceeds maximum limit for a numeric type |
| **FloatingPointError** | Raised when a floating point calculation fails. |
| **ZeroDivisionError** | Raised when division or modulo by zero takes place for all numeric types. |
| **AssertionError** | Raised in case of failure of the Assert statement. |
| **AttributeError** | Raised in case of failure of attribute reference or assignment. |
| **EOFError** | Raised when there is no input from either the raw\_input() or input() function and the end of file is reached. |
| **ImportError** | Raised when an import statement fails. |
| **KeyboardInterrupt** | Raised when the user interrupts program execution, usually by pressing Ctrl+c |
| **LookupError** | Base class for all lookup errors. |
| **IndexError** | Raised when an index is not found in a sequence. |
| **KeyError** | Raised when the specified key is not found in the dictionary. |
| **NameError** | Raised when an identifier is not found in the local or global namespace. |
| **UnboundLocalError** | Raised when trying to access a local variable in a function or method but no value has been assigned to it. |
| **EnvironmentError** | Base class for all exceptions that occur outside the Python environment. |
| **IOError** | Raised when an input/ output operation fails, such as the print statement or the open() function when trying to open a file that does not exist. |
| **IOError** | Raised for operating system-related errors. |
| **SyntaxError** | Raised when there is an error in Python syntax. |
| **IndentationError** | Raised when indentation is not specified properly. |
| **SystemError** | Raised when the interpreter finds an internal problem, but when this error is encountered the Python interpreter does not exit. |
| **SystemExit** | Raised when Python interpreter is quit by using the sys.exit() function. If not handled in the code, causes the interpreter to exit. |
| **TypeError** | Raised when an operation or function is attempted that is invalid for the specified data type. |
| **ValueError** | Raised when the built-in function for a data type has the valid type of arguments, but the arguments have invalid values specified. |
| **RuntimeError** | Raised when a generated error does not fall into any category. |
| **NotImplementedError** | Raised when an abstract method that needs to be implemented in an inherited class is not actually implemented. |

**Fallunterscheidungen**

if expression1:

statement(s)

elif expression2:

statement(s)

elif expression3:

statement(s)

else:

statement(s)

**Schleifen**

|  |  |
| --- | --- |
| *while loop*  Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body. | Example:  count = 0  while (count < 9):  print 'The count is:', count  count = count + 1  print "Good bye!" |
| *for loop*  Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.  Often in combination with range(start, stop, step) | Example:  fruits = ['banana', 'apple', 'mango']  for fruit in **fruits**: # Second Example  print 'Current fruit :', fruit |

**break statement:** terminates the current loop

**continue statement**: The **continue** statement rejects all the remaining statements in the current iteration of the loop and moves the control back to the top of the loop

**pass statement**: It is used when a statement is required syntactically but you do not want any command or code to execute.

**Comprehensions**

|  |  |
| --- | --- |
| list comprehension | [ expression for item in list if conditional ]  is equivalent to  for item in list:  if conditional:  expression |
| dictionary comprehension | { key : value for ( key, value ) in dictionary.items() if conditional } |

**Functions**