WWU
MÜNSTER

Python in GIS

Introduction: Python basics

living.knowledge

ifgi
Institut für Geoinformatik
Universität Münster

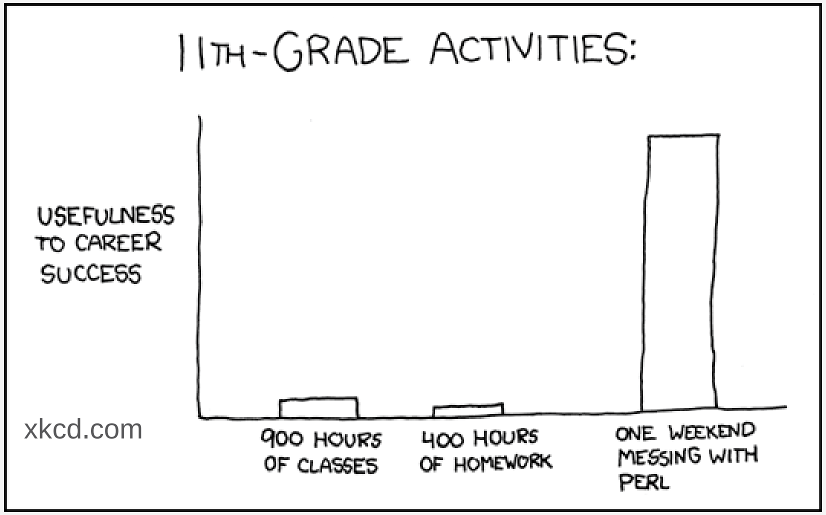
Learning goals

After this lesson you should be able to:

- Run a Python script
- Know the difference between a variable and a value
- Write functions in Python
- Formulate conditional statements
- Create for- and while-loops
- Handle strings, and lists
- Read and write files
- Know the types of errors in Python
- Use libraries, specifically OS and NumPy

Why learn programming?

- Structuring your work
- Repeatable and fast
- Separate source data and 'working data' - automatic conversion by a program!
- Developing models, websites, ...



xkcd.com

WWU
MÜNSTER

Why Python?

- High-level language easier to learn
- Free and open source software
- Highly scalable (used for both small, simple scripts, and large, complex projects)
- Runs on all platforms (i.e. Microsoft Windows, Linux, Unix, Apple Macintosh)
- Comes with many modules/libraries (pre-programmed functions)
- Common in the GIS world

Website and software: http://www.python.org

Python distributions (1)

Python versions are independent

Can install multiple versions on one machine

Python 2 and 3 are different, not backwards compatible

So, think about which version to use, specifically between 2 and 3, when writing a script

Python distributions (2)

There are three ‘types’:

1. The official stand-alone Python distribution, available from python.org
2. Other Python distribution packages, with editors and libraries, such as:
   - **Anaconda**
   - **Enthought’s Canopy**
   - **Python(x,y) (Spyder)**
3. Python embedded in other software, such as:
   - **ArcGIS**
   - **QGIS**

Creating and running a Python script

- A python program is an ascii file
- It can be created or edited with any text editor (e.g. vi, Wordpad, Notepad++ etc)
- You can also use editors specifically for Python (e.g. IDLE, Canopy, Spyder)

Executing a python program:

- type on the command prompt:

```
python myProgram.py
```

- or use the ‘Run’ button in a dedicated editor

All statements will be executed from top to bottom!

Python distributions in this workshop

We ONLY use the Python 3 distributions coming with:

- QGIS 3
- ArcGIS Pro

Advantages:

- You do not have to install Python or basic libraries
- You do not have to set environment variables
- You can use GIS libraries

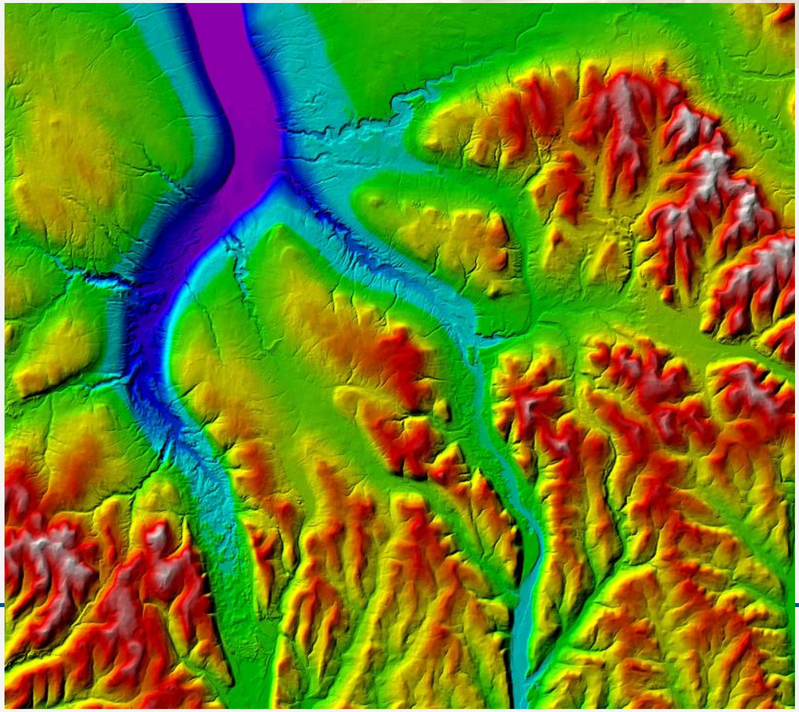
Disadvantages:

- You cannot use ArcGIS and QGIS tools within one script
- It is not easy to install additional libraries

Case study: data

We use:

- a GPS track
- a digital elevation model (DEM)



Variables, expressions and functions

Types of values

Values belong to a type. The most important built-in types are:

String:

- names, text printed on the screen or written to a file
- Examples: "This is a string", or "0.234", or " " (whitespace)

Integer (discrete)

- categories, counters (e.g., 0,1,2,3,4...100)
- Examples: 2, or 3, or -2, or 0, not 0.0!

Floating-point (continuous)

- scalar values used in calculations
- Examples: 2.234, or -12.3234, or 2343.1, or 0.0

Boolean

- result of comparisons: 0 (FALSE) or 1 (TRUE)

Variables (1)

Often, we do not use values directly in a script.

A variable is a way to reference to a known or unknown value

Assigning a value to a variable:

- streamPower = 23.4
- myName = "Piet“

Question: What are the types of these two variables?

Why handy?

- define all model inputs in one location
- can change during run time, for example ‘age’

Variables (2)

Meaning of “=“ is

- equality in mathematics
- assignment in Python, assigning a value to a variable

Equality in Python is “==“

This will be discussed later.

Variables (3)

Some rules:

- use meaningful names
- no spaces
- first letter a lowercase

e.g.,

```
streamPower
stream_power

instead of:

StreamPower
stream Power
```

Expressions

An expression is an instruction to execute something

A simple program (saved as python1_simple.py):

python1_simple.py

```
rectangle_length = 12.5
rectangle_width = 3.0
rectangle_area = rectangle_length * rectangle_width
print("The area of the rectangle is: ")
print(rectangle_area)

```

Python Console

```
Python Console
Use iface to access QGIS API interface
>>> exec(open('C:/Users/verstege/Docum
2018/scripts/python1_simple.py'.encode
The area of the rectangle is:
37.5
```

Question: What does this program do?

Functions, syntax

Syntax:

rV1, rV2,..,rVn = functionName(*arg1, arg2,..,argm*)

with:

- *rV1, rV2,..,rVn*: return values 1..*n*
- *arg1, arg2,..,argm*: arguments 1..*m*
- functionName: the name of the function

The function 'reads' the inputs (arguments), does 'something' and assigns the returned values to its outputs, the variables.

Using built-in functions

The **built-in function** float reads the value of the argument, converts it to a floating-point and returns a floating-point value:

```
# making a float
an_integer=2
a_floating_point=float(an_integer)
```

A hashtag (#) makes that the expression after it is ignored by Python.

Can be used to:

- put comments in the script (do this!)
- (temporarily) comment out parts of the script, e.g. when testing

Creating functions

You can also create functions yourself:

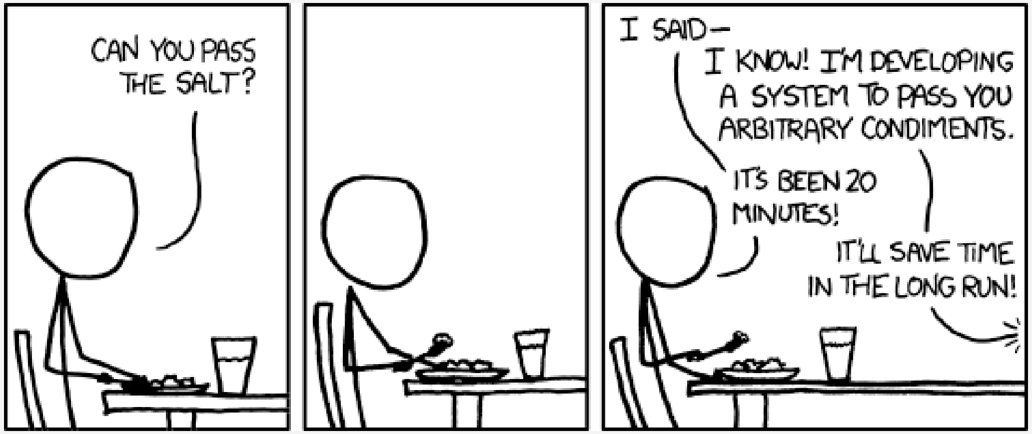
- new functions are built as a combination of existing python components (expressions)
- the definition of a new function is given in the main program or in an associated file
- a new function can be used anywhere in the program

Question: Why would you want to create functions?

Why creating functions?

- You can group statements that serve one purpose; this makes the program easier to read and to debug
- Functions make the script shorter by eliminating repetitive code.
- If you want to change something in the function you only have to do it once, in the repetitive code this would be several times
- Functions can be reused by others or in other programs of your own

Encapsulation



Function definition, syntax

def *functionName*(*arg1*,*arg2*,..,*argn*):

statement1

..

statementm

return *varReturn1*, *varReturn2*,..,*varReturnl*

with:

- *functionName*: the name of the new function
- *arg1*, *arg2*, ..., *argn*: input arguments
- *statement1*, ..., *statementm*: expressions doing something with the inputs
- *varReturn1*, *varReturn2*,...,*varReturnl*: variables returned by the function

Function definition, example

The function calculate_rectangle_area() with two input arguments returns one value:

python2_function.py

```python
def calculate_rectangle_area(width,length):
    rectangle_area = width*length
    return rectangle_area

rectangle_length = 12.5
rectangle_width = 3.0
area=calculate_rectangle_area(rectangle_length,\
    rectangle_width)
print("The area of the rectangle is: ", area)
```

Python Console

```
1 Python Console
2 Use iface to access QGIS API interface or Typ
3 >>> exec(open('C:/Users/verstege/Documents/ec
  2018/scripts/python2_function.py'.encode('utf
4 The area of the rectangle is:  37.5
5
```

Indentation is critical in Python!

Function definition, example

A variable created in a function does not exist outside the function! E.g.:

python2_function_wrong.py

```python
def calculate_rectangle_area(width,length):
    rectangle_area = width*length
    return rectangle_area

rectangle_length = 12.5
rectangle_width = 3.0
area = calculate_rectangle_area(rectangle_length,\
    rectangle_width)
print("The area of the rectangle is: ", rectangle_area)
```

```
Traceback (most recent call last):
  File "C:\Program Files\QGIS 3.0\apps\Python36\Lib\code.py",
n runcode
    exec(code, self.locals)
  File "<input>", line 1, in <module>
  File "<string>", line 10, in <module>
NameError: name 'rectangle_area' is not defined
```

Conditionals

Comparison operators

Comparison operators compare two values or, more commonly, variables

```
x == y    # TRUE if x is equal to y
x != y    # TRUE if x is not equal to y
x > y     # TRUE if x is greater than y
x < y     # TRUE if x is less than y
x >= y    # TRUE if x is greater than or equal to y
x <= y    # TRUE if x is less than or equal to y
```

The result of comparison operators is a 0 (FALSE) or 1 (TRUE), of type Boolean.

Comparison operators

The result of comparison operators is a 0 (FALSE) or 1 (TRUE), of type Boolean.

```
a = 4>3
print(a)
print(type(a))
```

```
1
<type 'bool'>
```

Logical operators

Logical operators evaluate the logical relation between two values or variables

```
x and y   # TRUE if both x and y are TRUE
x or y    # TRUE if x or y are TRUE
not x     # TRUE if x is FALSE
```

The operands (x and y above) are in most cases Booleans where:

a 0 is considered False

a value unequal to 0 is considered True

The result of logical operators is a 0 (False) or 1 (True), of type Boolean.

Combine comparison and logical operators

For instance:

```
(a >= b) and not (d < c)

(2*a < 100.0) or (b/3 > c)
```

Conditional statements, syntax

A conditional statement checks whether a condition is fulfilled and only if it does, it executes a block of code.

```
if CONDITION:
  STATEMENT1
  ...
  STATEMENTn
```

with:

- CONDITION, an expression with a Boolean result
- STATEMENT1,..,STATMENTn, expressions executed if the CONDITION is TRUE

Conditional statements, example (1)

For instance:

```
if (rain > 0):
    print("stay at home!")
```

Cond. statements and alternatives, syntax

A conditional statement checks whether a condition is fulfilled and only if it does, it executes a block of code.

You can also define a block of code that is executed if the condition is **not** fulfilled:

```
if CONDITION:
  STATEMENT1
  ...
  STATEMENTn
else:
  ALTSTAT1
  ..
  ALTSTATm
```

with:

- ALTSTAT1..ALTSTATm, expressions executed if CONDITION is FALSE

Conditional statements, example (1a)

Our previous example:

```
if (rain > 0):
   print("stay at home!")
else:
   print("go swimming!")
```

Conditional statements chained, syntax

You can also chain different conditional statements. The second is checked if the first is not fulfilled.

```
if CONDITION:
 STATEMENT1
 ...
 STATEMENTn
elif ANOTHERCOND:
 ALTSTAT1
 ..
 ALTSTATm
else:
 ALTALTSTAT1
 ..
 ALTALTSTATl
```

Conditional statements, example (1b)

Our previous example:

```
if (rain > 0):
   print("stay at home!")
elif (temperature > 30):
   print("go swimming!")
else:
   print("have a drink!“)
```

Exercise #1

- Create an input variable ‘choice’ and assign a Boolean value to it
- Write a function that returns ‘the user said yes’ when it receives a Boolean True as input and ‘the user said no’ when it receives a Boolean False as input
- Test your function with the variable ‘choice’ and print the result

How to create and run a Python script:

- Create an <u>empty text file</u> in the folder where you want to work
- (Re)name it (to) e.g. ex1.py
- Right-click the new file and choose <u>Edit with IDLE (ArcGIS Pro)</u>
- Add your code
- Run it with <u>(Fn) F5</u> or from the menu: <u>Run → Run Module</u>

Loops

Loops, the for statement, syntax

The for statement is used in loops when you already know in advance how many iterations are needed.

```
for ELEMENT in COMPOUND:
  STATEMENT1
  ...
  STATEMENTn
```

with

- ELEMENT, an element which can be of any type
- COMPOUND, a compound data type, e.g. a list (explained later)
- STATEMENT1,..,STATEMENTn, expressions in the loop

For statement, example (1)

python3_listfor.py

```python
a_list = ["New York", "Amsterdam", "Paris", "Rome"]

for i in a_list:
    print(i)
```

Python Console

```
Python Console
Use iface to access QGIS API interface or Type help(iface)
>>> exec(open('C:/Users/verstege/Documents/education/python_
2018/scripts/python3_listfor.py'.encode('utf-8')).read())
New York
Amsterdam
Paris
Rome
```

WWU
MÜNSTER

For statement, example (2)

python4_rangefor.py

```python
for number in range(1,5):
    print(number)
```

Python Console

```
Python Console
Use iface to access QGIS API interface or Type help(iface)
>>> exec(open('C:/Users/verstege/Documents/education/python
2018/scripts/python4_rangefor.py'.encode('utf-8')).read())
1
2
3
4
```

Loops, the while statement, syntax

The while statements is used for loops when you do **not** know how many iterations are needed.

```
while CONDITION:
  STATEMENT1
  ...
  STATEMENTn
```

with:

- CONDITION, a Boolean expression
- STATEMENT1,..,STATEMENTn, the statements in the loop
- Note: STATEMENT1,..,STATEMENTn generally determine CONDITION

Loops, the while statement, example (1)

```
# program with a while loop
n = 0
while n < 20:
   print(n, end=" ")
   n = n+1
```

Question: What does ‘end’ do?

Question: What does this print?

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
```

Loops, the while statement, example (2)

```
# program with a while loop
n = 0
while n < 20:
   print(n, end=" ")
   n = n+1
print("The value of n after the loop is:", n)
```

Question: What does this print?

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
The value after the loop is: 20
```

Loops, the while statement, example (3)

Change into:

```
# program with a while loop
n = 40
while n < 20:
    print(n, end=" ")
    n = n+1
print("The value of n after the loop is:", n)
```

Question: What does this print?

```
The value after the loop is: 40
```

Loops, the while statement, example (4)

```
# program with a while loop
n = 0
while 1:
   print(n, end=" ")
   n = n+1
print("The value of n after the loop is:", n)
```

Question: What does this print?

Local variables (1)

Variables created in a function are local variables:

→ they are not known outside the function

E.g. this program:

```
def aFunction():
    n = 0

aFunction()
print(n)
```

```
Traceback (most recent call last):
  File "local0.py", line 5, in ?
    print(n)
NameError: name 'n' is not defined
```

Local variables (2)

Variables created in a function are local variables:

→ they are not known outside the function

→ they do not affect variables outside the function

E.g. this program:

```
def aFunction():
    n = 0
    print("n inside the function:", n)

n = 100
aFunction()
print("n outside the function:", n)
```

```
n inside the function: 0
n outside the function: 100
```

Local variables (3)

Variables in a <u>loop</u> are NOT local variables:

E.g. this program:

```
n = 0
while n < 10:
    n = n+1

print n
```

```
10
```

Strings

Compound data, syntax of bracket operator

Compound data type: data type consisting of smaller pieces

Data type string: compound data type consisting of letters

Selecting a single string with the bracket [] operator:

```
LETTER = STRING[J]
```

with:

- STRING, a variable of data type string
- J, index, a variable of data type integer
- LETTER, a letter of STRING (note: LETTER is also of type string)

Bracket operator, non-negative index

LETTER = STRING[J]

If J ≥ 0:

LETTER is the (J+1)-eth letter of STRING

So the first element has index zero!

Example:

python7_brackets.py

```
name = "Sandy"
firstLetter=name[0]
secondLetter=name[1]
lastLetter=name[4]
print(firstLetter, secondLetter, lastLetter)
```

Python Console

```
Python Console
Use iface to access QGIS API interface or Type help(iface) for more in
>>> exec(open('C:/Users/verstege/Documents/education/python_in_GIS/201
2018/scripts/python7_brackets.py'.encode('utf-8')).read())
S a y
```

Bracket operator, negative index

LETTER = STRING[J]

If J < 0:

J = -1 yields the last letter of STRING

J = -2 the letter before, etc.

Example:

python7_brackets_backwards.py

```python
name = "Sandy"
firstLetter=name[-5]
secondLetter=name[-4]
lastLetter=name[-1]
print(firstLetter, secondLetter, lastLetter)
```

Python Console

```
Python Console
Use iface to access QGIS API interface or Type help(iface) for more in
>>> exec(open('C:/Users/verstege/Documents/education/python_in_GIS/201
2018/scripts/python7_brackets.py'.encode('utf-8')).read())
S a y
```

Compound data, syntax of bracket operator

String slice: a segment of a string

Syntax:

SLICE = STRING[I:J]

with:

- STRING, a variable of data type string
- I, index for start of segment, a variable of data type integer
- J, index for end of segment, a variable of data type integer
- SLICE, a segment of STRING (note: SLICE is also of type string)

WWU MÜNSTER

Bracket operator, slices (1)

SLICE = STRING[I:J]

I and J non-negative, J should be greater than I:

SLICE consists of the (I+1)-eth up to and including the J-eth character

Example:

python7_brackets_slice.py

```python
name = "Sandy"
firstSlice=name[0:3]
secondSlice=name[1:3]
print(firstSlice, secondSlice)

```

Python Console

```
Python Console
Use iface to access QGIS API interface or Type help(i
>>> exec(open('C:/Users/verstege/Documents/education/
2018/scripts/python7_brackets_slice.py'.encode('utf-8
San an

```

Bracket operator, slices (2)

SLICE = STRING[I:J]

Omitting I: the slice starts at the beginning of STRING

Omitting J: the slice goes to the end of STRING

Example:

python7_brackets_slice_omit.py

```python
name = "Sandy"
firstSlice=name[:3]
secondSlice=name[1:]
print(firstSlice, secondSlice)
```

Python Console

```
Python Console
Use iface to access QGIS API interface or Type
>>> exec(open('C:/Users/verstege/Documents/edu
2018/scripts/python7_brackets_slice_omit.py'.e
San andy
```

Bracket operator, example (1)

Given: a variable that contains the name of file:

```
filename="data.col"
```

Aim: a program that prints just the basename of the filename

```
data
```

WWU
MÜNSTER

Bracket operator, example (2)

Let us start with a script that prints all letters until it encounters a dot:

Python Console

```
Python Console
Use iface to access QGIS API interface or Type
>>> exec(open('C:/Users/verstege/Documents/educ
2018/scripts/python8_data1.py'.encode('utf-8'))
d
a
t
a
found a dot!
```

python8_data1.py

```python
filename="data.col"

for letter in filename:
    if letter == ".":
        print("found a dot!")
        break
    print(letter)
```

Bracket operator, example (3)

python8_data2.py

```python
filename="data.col"
basename = ""

for letter in filename:
    if letter == ".":
        print("found a dot!")
        break
    basename += letter
    print(basename)
```

```
ess QGIS API interface or Type help(
>>> exec(open('C:/Users/verstege/Documents/education/
2018/scripts/python8_data2.py'.encode('utf-8')).read
d
da
dat
data
found a dot!
```

Bracket operator, example (4)

python8_data2.py

```
filename="data.col"
basename = ""

for letter in filename:
    if letter == ".":
##      print("found a dot!")
        break
    basename += letter

print(basename)
```

Python Console

```
Python Console
Use iface to access QGIS API interface or Type
>>> exec(open('C:/Users/verstege/Documents/edu
2018/scripts/python8_data2.py'.encode('utf-8')
data
```

Lists

What is a list?

Ordered set of values, values are the so-called elements of a list

An element can be 'anything', e.g.

- a string
- a floating-point
- another list
- etc.

Each element is identified by an index

Comparison between strings and lists

Resemblances:

- both are compound data types, consisting of elements
- both refer to an element using an index
- both use bracket operator “[ ]” for referring to elements

Difference:

- string elements are single letters (of type string); list elements can be anything

Creating lists

Most often used are:

```
first_list = [0.12, 23.4, 12.5]                  # three elements
                                                 # of type floating-point

second_list = ["New York", "Amsterdam"]          # two elements
                                                 # of type string

third_list = [3, 5, 7, 9]                        # four elements
                                                 # of type integer
```

The third_list can also be created with the range function:

```
third_list = range(3, 10, 2)                     # the list [3, 5, 7, 9]
```

Accessing single elements

Use bracket operator

Very similar to accessing elements of a string

python9_list1.py

```python
a_string="New York"
print(a_string[0])

a_list = ["New York", "Amsterdam", "Paris", "Rome",\
              "Berlin", "Madrid"]
print(a_list[0])

```

```
Use iface to access QGIS API interface or Type help(ifa
>>> exec(open('C:/Users/verstege/Documents/education/py
2018/scripts/python9_list1.py'.encode('utf-8')).read()
N
New York
```

Accessing slices

Use bracket operator

Very similar to accessing slices of a string

python9_list2.py

```
a_string="New York"
print(a_string[1:5])

a_list = ["New York", "Amsterdam", "Paris", "Rome",\
            "Berlin", "Madrid"]
print(a_list[1:5])
```

```
Use iface to access QGIS API interface or Type help(i
>>> exec(open('C:/Users/verstege/Documents/education/
2018/scripts/python9_list2.py'.encode('utf-8')).read(
ew Y
['Amsterdam', 'Paris', 'Rome', 'Berlin']
```

Accessing elements in a loop (1)

With a for loop (shortest):

python3_listfor.py

```python
a_list = ["New York", "Amsterdam", "Paris", "Rome"]

for i in a_list:
    print(i)
```

Python Console

```
Python Console
Use iface to access QGIS API interface or Type help(iface
>>> exec(open('C:/Users/verstege/Documents/education/pyt
2018/scripts/python3_listfor.py'.encode('utf-8')).read()
New York
Amsterdam
Paris
Rome
```

Accessing elements in a loop (2)

With a while loop:

python9_list3.py

```python
a_list = ["New York", "Amsterdam", "Paris", "Rome"]
i = 0
while i < len(a_list):
    print(a_list[i])
    i = i+1
```

Python Console

```
Python Console
Use iface to access QGIS API interface or Type help(iface
>>> exec(open('C:/Users/verstege/Documents/education/pyth
2018/scripts/python3_listfor.py'.encode('utf-8')).read()
New York
Amsterdam
Paris
Rome
```

Strings are immutable, lists are mutable (1)

Strings are **immutable**, i.e. you cannot directly change an element:

```
a_string = "Back"
# try to change the "B" to a "J"
a_string[0]="J"
```

prints:

```
Traceback (most recent call last):
  File "stringmutable.py", line 3, in ?
    a_string[0]="J"
TypeError: object doesn't support item assignment
```

Strings are immutable, lists are mutable (2)

Lists are mutable, i.e. you can directly change an element:

```
aList = [0.12, 23.4, 12.5]
# change the first element (0.12) to 2.34
aList[0]=2.34
print aList
```

prints:

```
[2.34, 23.4, 12.5]
```

Strings are unmutable, lists are mutable (3)

Updating slices of a list:

python9_list4.py

```python
a_list = [1, 2, 3, 4, 5, 6]

a_list[1:4]=[9]     # replace elements with one element
print(a_list)

a_list[0:0]=[0,0]   # insert two elements with value 0
print(a_list)

a_list[1:3]=[]      # delete two elements
print(a_list)
```

```
p(iface) f
on/python_
2018/scripts/python9_list4.py'.encode('utf-8')).read())
[1, 9, 5, 6]
[0, 0, 1, 9, 5, 6]
[0, 9, 5, 6]
```

Nested lists

A list that is an element in another list, e.g,:

```
samples = [["x","y","z"],[12,32,7],[12,40,7]]
```

All combinations of lengths and types are possible, e.g.:

```
a_list = [14.2,[12,32],[12,40,"peter"]]
```

Accessing an element in a nested list

Syntax corresponds to 'normal' lists, e.g.:

python9_list5.py

```
samples = [14.2,[12,32],[12,40,"peter"]]

third = samples[2]
print(third[1])

print(samples[2][1])

```

```
Python Console
Use iface to access QGIS API interface or Type help(iface) for m
>>> exec(open('C:/Users/verstege/Documents/education/python_in_G
2018/scripts/python9_list5.py'.encode('utf-8')).read())
40
40

```

Tuples and dictionaries in short

Tuples are lists that are immutable. They are defined using normal brackets ‘()’ instead of square brackets ‘[]’.

e.g.

```
>>> a_tuple = (0,1,2)
```

Dictionaries are key-value pairs. In other languages, sometimes found as ‘associative memories’. They are defined using curly brackets.

e.g.

```
>>> days = {‘Monday’: ‘Montag’, ‘Tuesday’: ‘Dienstag’}
>>> days[‘Monday’]
‘Montag’
>>> days.get(‘Monday’)
‘Montag’
>>> list(days.keys())
 [‘Monday’, ‘Tuesday’]
```

Exercise #2

We have a nested list:

```
samples = [["x","y","z"],[12,32,7],[12,40,7]]
```

Make a program in Python that prints each individual value, formatted as a table:

```
 x     y    z
12    32    7
12    40    7
```

Files

Files (1)

Computer memory

- used by the program to store data (e.g. variables) while running
- disappears when the program ends or the computer shuts down is mainly managed by Python (you don't need to do that)

Files

- can be used in a program to load or store specified data
- storage and manipulation needs to be defined in the program (explicitly)

Files (2)

Like with a book, you need to do the following steps to read/write from/to a file:

- open the file
- read from the file OR write to the file
- close the file

The first thing you'll need to do is use Python's built-in open() function to get a ***file object***.

```
>>> f = open('name.txt', 'w')
```

Use 'r' for read, 'w' for write, and 'a' for append.

Files (3)

It is good practice to use the with keyword when dealing with file objects.

Then the file is properly closed after its suite finishes, even if an exception is raised.

```
with open('workfile') as f:
    data = f.read()
```

If you’re not using the with keyword, then you should call f.close() to close the file:

```
f = open(‘name.txt’, ‘w’)
f.close()
```

Error handling

Types of Python errors

A Python script terminates as soon as it encounters an error.

In Python, an error can be:

1. a **syntax error** = incorrect statement (e.g. closing bracket missing)

python10_syntax_error.py

```
samples = [14.2,[12,32],[12,40,"peter"]]

third = samples[2]
print(third[1])

print(samples[2][1]))
```

invalid syntax

2. an **exception** = a **runtime error** = code is syntactically correct, but a problem is encountered while the script running

Built-in exceptions (1)

Exception	Description
ImportError	Raised when the imported module is not found.
IndexError	Raised when the index of a sequence is out of range.
MemoryError	Raised when an operation runs out of memory.
RuntimeError	Raised when an error does not fall under any other category.
IndentationError	Raised when there is an incorrect indentation.
ZeroDivisionError	Raised when the second operand of a division is zero.

See: https://docs.python.org/3/library/exceptions.html

Built-in exceptions ()

Example of a ZeroDivisionError:

```
Python Console
Use iface to access QGIS API interface or Type help(iface) for more info
Security warning: typing commands from an untrusted source can lead to data loss and/or leak
>>> exec(open('C:/Users/verstege/Documents/education/python_in_GIS/2018_2019/scripts/2_python/python10_ZeroDivisionError.py'.encode('utf-8')).read())
Traceback (most recent call last):
  File "C:\Program Files\QGIS 3.4\apps\Python37\Lib\code.py", line 90, in runcode
    exec(code, self.locals)
  File "<input>", line 1, in <module>
  File "<string>", line 1, in <module>
ZeroDivisionError: division by zero
```

python10_ZeroDivisionError.py

```python
print(0/0)
```

WWU MÜNSTER

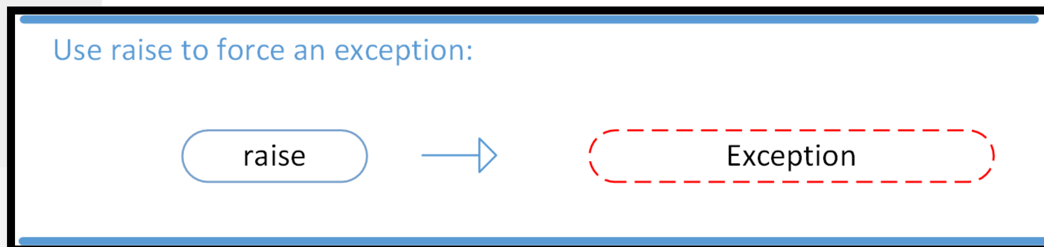
Self-defined exceptions

We can use raise to throw a built-in or a self-defined exception if a condition occurs.

```
Python Console
Use iface to access QGIS API interface or Type help(iface) for more info
Security warning: typing commands from an untrusted source can lead to data loss and/or leak
>>> exec(open('C:/Users/verstege/Documents/education/python_in_GIS/2018_2019/scripts/2_python/python10_raise.py'.encode('utf-8')).read())
Traceback (most recent call last):
  File "C:\Program Files\QGIS 3.4\apps\Python37\Lib\code.py", line 90, in runcode
    exec(code, self.locals)
  File "<input>", line 1, in <module>
  File "<string>", line 4, in <module>
Exception: x should not exceed 5.     The value of x was: 10
```

python10_raise.py

```python
x = 10
if x > 5:
    raise Exception('x should not exceed 5. \
    The value of x was: ' + str(x))
```

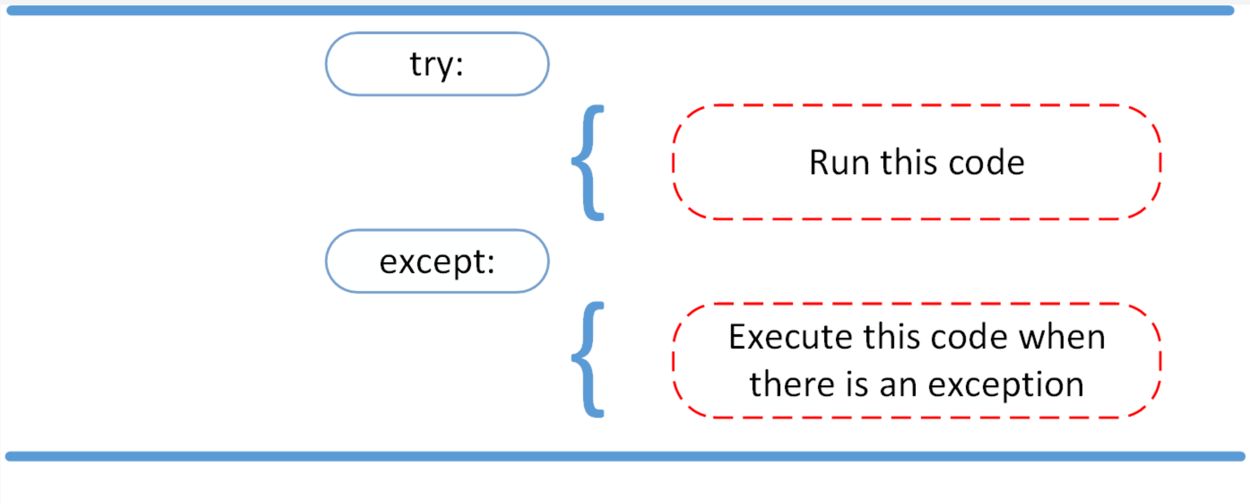


Try ... Except blocks (1)

The try and except block in Python is used to catch and handle exceptions.

The code following the try statement is handled as a “normal” part of the code.

The code following the except statement is the response to any exceptions in the preceding try clause.



Everything after that is run, despite potential exceptions!

Try ... Except blocks (2)

Example:

```
Python Console
Use iface to access QGIS API interface or Type help(iface) for more info
Security warning: typing commands from an untrusted source can lead to data loss and/or leak
>>> exec(open('C:/Users/verstege/Documents/education/python_in_GIS/2018_2019/scripts/2_python/python10_try_except.py'.encode('utf-8')).read())
Could not open file.log
Hello
```

python10_try_except.py

```python
try:
    with open('file.log') as file:
        read_data = file.read()
except:
    print('Could not open file.log')

print("Hello")
```

Libraries

What is a library?

What you typically see at the top of a Python script are import statements

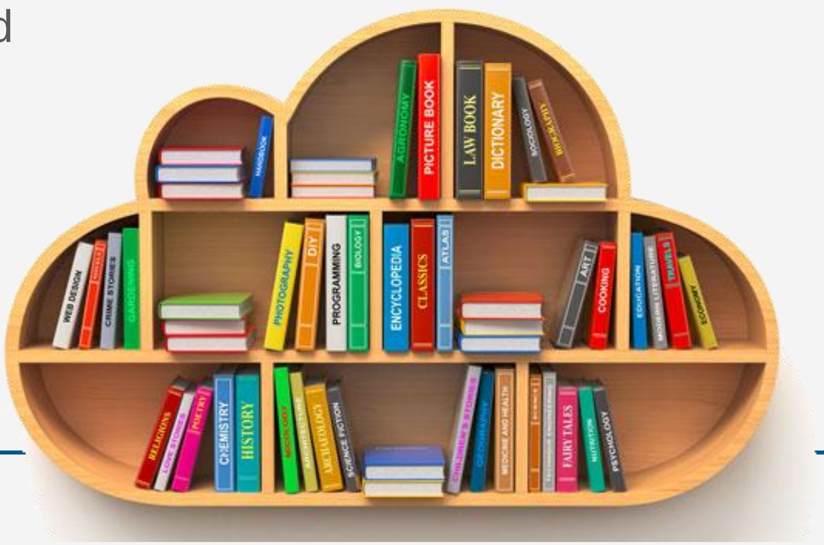
These lines of code load additional libraries, also called modules

A library is a package of Python code you can access and use from scripts

It is possible to make your own libraries

Now, I’ll introduce general, commonly-used libraries

Later on, GIS-specific libraries are introduced



Modules/libraries

A module is a file with a collection of related functions. It needs to be imported at the top of a program, e.g.:

```
import string
import math
```

Functions from a module are called using dot notation, e.g.:

```
a_new_name=string.replace(a_name, "te", "tra")
log_value=math.log10(value)
```

string library

```
import string

a_string ="sandY"

capitalize= a_string.capitalize()        # returns Sandy (a string)
lower= a_string.lower()                  # returns sandy (a string)
replace= a_string.replace("sa","ci")     # returns cindY (a string)
find= a_string.find(,"n")                # returns 2 (an integer),
                                         # index of the letter n
```

WWU MÜNSTER

Remember this script?

python8_data2.py

```
filename="data.col"
basename = ""

for letter in filename:
    if letter == ".":
##        print("found a dot!")
        break
    basename += letter

print(basename)
```

Python Console

```
Python Console
Use iface to access QGIS API interface or Type help(iface)
>>> exec(open('C:/Users/verstege/Documents/education/pytho
2018/scripts/python8_data2.py'.encode('utf-8')).read())
data
```

This script can be simplified with the string module, method 1:

basic_libs7_string.py

```
import string

# method 1
filename = "data.col"
index_dot = filename.find(".")
print(filename[0:index_dot])
```

WWU MÜNSTER

This script can be simplified with the string module, method 2:

basic_libs7_string.py

```
import string

# method 1
filename = "data.col"
index_dot = filename.find(".")
print(filename[0:index_dot])

# method 2
parts = filename.split(".")
print(parts[0])
```

```
Use iface to access QGIS API interface or Type hel
>>> exec(open('C:/Users/verstege/Documents/educati
2018/scripts/basic_libs7_string.py'.encode('utf-8
data
data
```

os library

Provides a portable way of using operating system dependent functionality

Very handy: `os.path`

`os.path.exists()` checks if a pathname exists

Watch out with backslashes!

basic_libs2.py

```python
# Import the os module for the next couple of Windows examples
import os

# Backslashes in Windows filenames
print(os.path.exists('c:\Users\verstege\Documents\temp'))
```

(unicode error) 'unicodeescape' codec can't decode bytes in position 2-3: truncated \UXXXXXXXX escape

os - paths (1)

Escape Sequence	Meaning
newline	Ignored
\\	Backslash (\)
\'	Single quote (')
\"	Double quote (")
\a	ASCII Bell (BEL)
\b	ASCII Backspace (BS)
\f	ASCII Formfeed (FF)
\n	ASCII Linefeed (LF)
\r	ASCII Carriage Return (CR)
\t	ASCII Horizontal Tab (TAB)
\v	ASCII Vertical Tab (VT)
ooo	ASCII character with octal value *ooo*
\x*hh...*	ASCII character with hex value *hh...*

os - paths (2)

basic_libs2.py

```python

# Three ways to fix the problem
# Use forward slashes instead
print(os.path.exists('c:/Users/verstege/Documents/temp'))
# Use double-backslashes
print(os.path.exists('c:\\Users\\verstege\\Documents\\temp'))
# Prefix the string with r
print(os.path.exists(r'c:\Users\verstege\Documents\temp'))

# OR (less error prone)
# Use the function join()
# But you still need backslashes
path = os.path.join('c:\\', 'Users', 'verstege',
                    'Documents', 'temp')
print(os.path.exists(path))
os.chdir(path)
```

os - paths (3)

```python
# Or start defining the path from the current directory
# (of the script or of the Python distribution!!)
print(os.getcwd())
os.chdir(os.path.join(os.getcwd(), 'calibration'))
print(os.getcwd())

```

Python Console

```
Python Console
Use iface to access QGIS API interface or Type help(iface) for more info
>>> exec(open('C:/Users/verstege/Documents/education/python_in_GIS/2017_2018/scripts/basic_libs2.py'.encode('utf-8')).read())
True
True
True
True
c:\Users\verstege\Documents\temp
c:\Users\verstege\Documents\temp\calibration

```

math library

Provides access to mathematical functions and commonly used constants.

Documentation: https://docs.python.org/2/library/math.html

basic_libs5.py

```python
##### math #####
import math

# special constants
print('pi:', math.pi)
print('e:', math.e)
```

Python Console

```
Python Console
Use iface to access QGIS API interface or
e info
>>> exec(open('C:/Users/verstege/Document
/2017_2018/scripts/basic_libs5.py'.encode
pi: 3.141592653589793
e: 2.718281828459045
```

math - power and logarithms

`math.exp(x)`

Return ex.**

`math.log(x[, base])`

With one argument, return the natural logarithm of x (to base e).

With two arguments, return the logarithm of x to the given base, calculated as log(x)/log(base).

`math.sqrt(x)`

Return the square root of x.

math - testing for exceptional values

`math.isinf(x)`

Check if the float *x* is positive or negative infinity.

`math.isnan(x)`

Check if the float *x* is a NaN (not a number).

When would you use this in a GIS context?

NumPy library

NumPy’s main object is the homogeneous multidimensional array:

- a table of elements
- all of the same type (usually numbers)
- indexed by a tuple of positive integers

Guess what we’re going to use this array for?

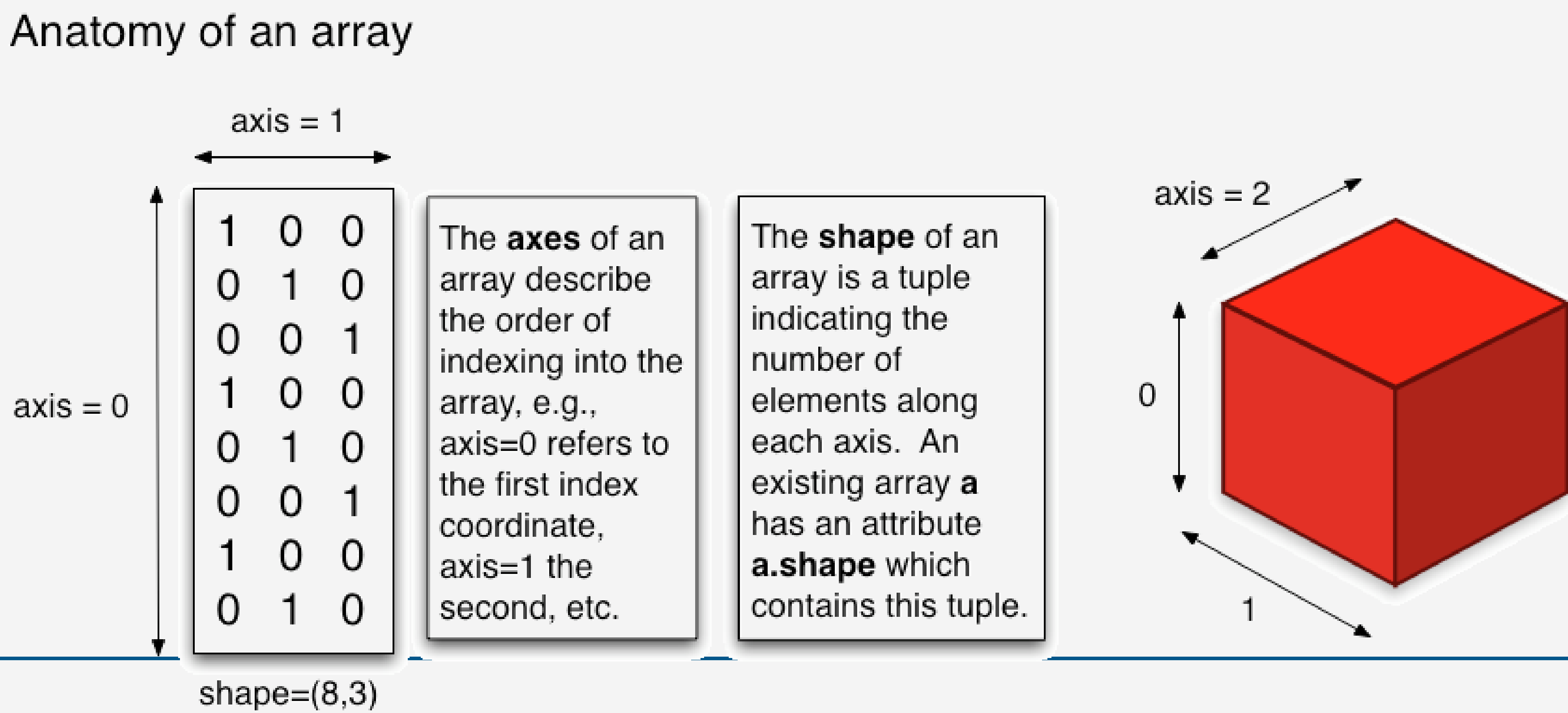
By convention, the numpy module is renamed to np when importing it

Documentation: www.numpy.org/

NumPy - ndarray (1)

NumPy's array class is called `ndarray`.

dimensions are called *axes*, the number of axes is *rank*.



NumPy - ndarray (2)

`ndarray.ndim` - number of axes (dimensions) of the array

`ndarray.shape` - the dimensions of the array, a tuple of integers indicating the size of the array in each dimension: with *n* rows and *m* columns (n,m)

`ndarray.size` - total number of elements of the array. This is equal to the product of the elements of shape.

`ndarray.dtype` - an object describing the type of the elements in the array.

`ndarray.itemsize` - the size in bytes of each element of the array.

`ndarray.data` - the buffer containing the actual elements of the array. Normally, we won’t need this attribute because we will access the elements in an array by indexing.

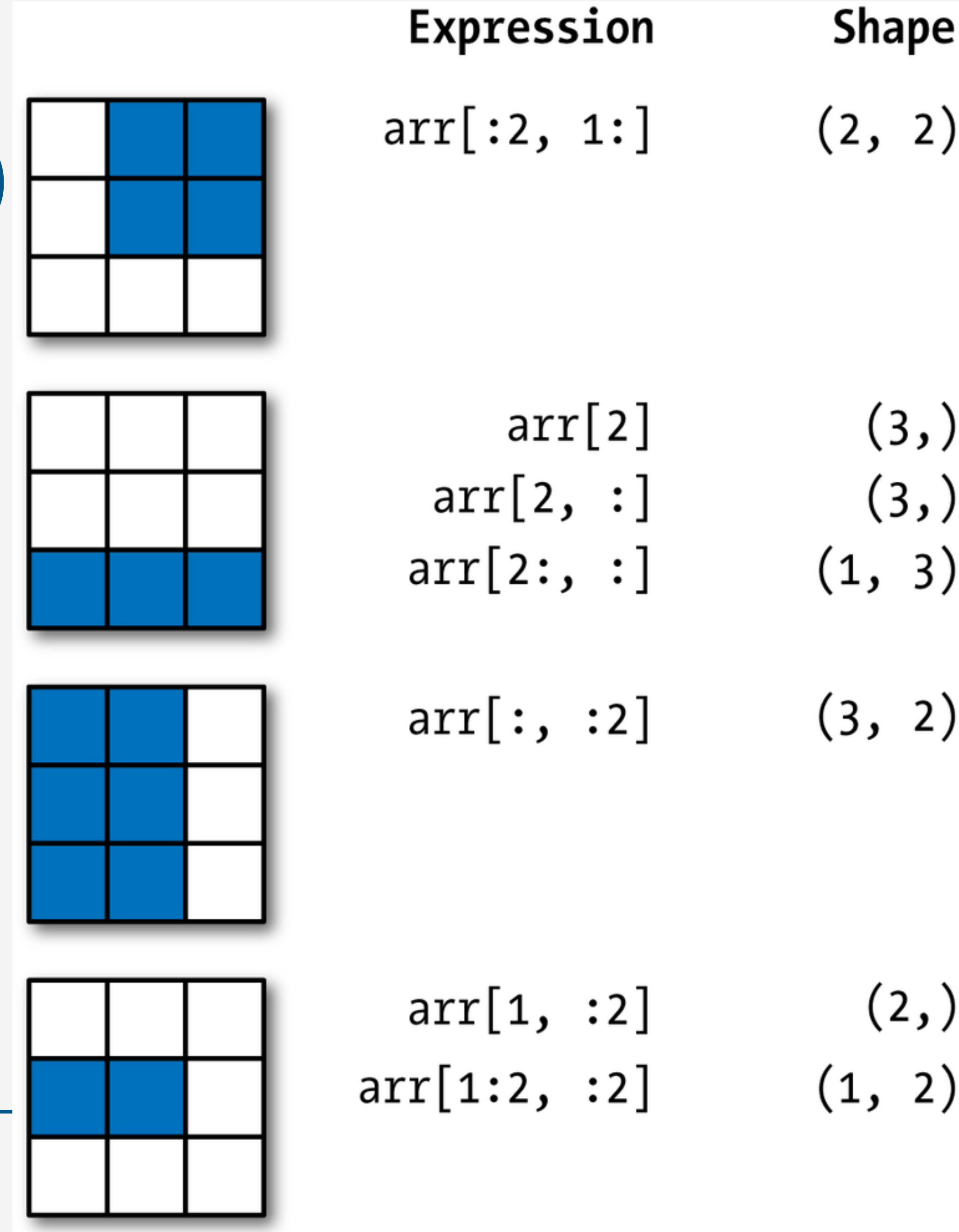
NumPy – slicing (1)

Same as with lists

But multidimensional

Separate by comma

Columns, rows!



WWU MÜNSTER

NumPy – slicing (2)

basic_libs6.py

```python
##### numpy #####
import numpy as np

a = np.array([[1, 3, 4], [2, 7, 6]])
b = np.array([[5, 2, 9], [3, 6, 4]])
print('a is', a)
print('b is', b)

# Slicing arrays
print(a[1])
# Like this: list of rows and list of columns
print('With indices:', a[[0,1],[1,2]])
# With Boolean array of the same shape
p = np.array([[True, True, False],
              [False, True, False]], dtype=bool)
print('Boolean way:', a[p])
```

Python Console

```
Python Console
Use iface to access QGIS AP
e info
>>> exec(open('C:/Users/ver
/2017_2018/scripts/basic_li
a is [[1 3 4]
 [2 7 6]]
b is [[5 2 9]
 [3 6 4]]
[2 7 6]
With indices: [3 6]
Boolean way: [1 3 7]
```

WWU MÜNSTER

NumPy - calculating or where (if, then, else)

basic_libs6.py

```python

# Calculating with arrays
print(a + b) 
print(a > b)

# Where, handy for selection
print(np.where(a > b, 10, 5))
print(np.where(a > b, a, b))
```

Python Console

```
[[ 6  5 13]
 [ 5 13 10]]
[[False  True False]
 [False  True  True]]
[[ 5 10  5]
 [ 5 10 10]]
[[5 3 9]
 [3 7 6]]
```

NumPy - creating standard arrays

basic_libs6_numpy.py

```python
# Create arrays.
print(np.zeros((3,2)))
print(np.ones((2,3), np.int))
print(np.ones((2,3), np.int) * 5)
print(np.empty((2,2)))

# check the data type
print(np.zeros((3,2)).dtype)
```

Python Console

```
[[0.  0.]
 [0.  0.]
 [0.  0.]]
[[1 1 1]
 [1 1 1]]
[[5 5 5]
 [5 5 5]]
[[4.24399158e-314 1.45993310e-311]
 [1.18575755e-322 1.68373916e-317]]
float64
```

WWU
MÜNSTER

NumPy - creating structured arrays (1)

Structured arrays can have a separate data type per column, i.e. attribute

And you can now refer to attributes either by index or by name (see next slide)

basic_libs6_numpy.py

```
# Use a compound data type for structured arrays
data = np.zeros(4, dtype={'names':('name', 'age', 'weight'),
                          'formats':('U10', 'i4', 'f8')})
print(data.dtype)
```

```
[('name', '<U10'), ('age', '<i4'), ('weight', '<f8')]
```

```
>>>
```


Documentation: https://docs.scipy.org/doc/numpy/user/basics.rec.html

NumPy - creating structured arrays (2)

basic_libs6_numpy.py

```python
# Now we can fill this structured array with data
# of the correct type
name = ['Alice', 'Bob', 'Cathy', 'Doug']
age = [25, 45, 37, 19]
weight = [55.0, 85.5, 68.0, 61.5]
data['name'] = name
data['age'] = age
data['weight'] = weight
print(data)
```

```
[('name', '<U10'), ('age', '<i4'), ('weight', '<f8')]
[('Alice', 25, 55. ) ('Bob', 45, 85.5) ('Cathy', 37, 68. )
 ('Doug', 19, 61.5)]

>>>
```

Exercise #3

- Create three input variables: r, location, and file_name
- Write a function that returns the area of a circle with a given radius (input)
- Let the script write **the input and result** of the function to a text file that is saved to disk in the folder defined by ‘location’ with the file name ‘file_name’