Argelander-
Institut
für
Astronomie

Python for Astronomers

Strings

Exercises - solutions

- Determine the smallest positive float of our Python installation by using a while-loop

```
>>> x = 1.
>>> while x>0:
...   x /= 2.
...   print x
```

Exercises - solutions

- Determine the numerical accuracy of the installation. Add smaller and smaller numbers to 1. and compare the result with 1.

```
>>> x = 1.
>>> while 1.+x>1.:
...   x /= 2.
...   print x
```

Exercises - solutions

- Determine the largest positive float of the installation. Import numpy and compare larger and larger numbers with numpy.inf

```
>>> # First test:
>>> x = 1.
>>> while 1:
...     x*= 2.
...     print x
... <Ctrl>-c
```

Exercises - solutions

```
>>> # inf indicates positive
>>> # infinity (IEEE 754: Floating
>>> # Point Arithmetic)

>>> import numpy
>>> x = 1.
>>> while x!=numpy.inf:
...   x *= 2.
...   print x
```

Alternative solutions

```
>>> x = 1.
>>> while x!=float('inf'):
...   x *= 2.
...   print x
```

Alternative solutions

```
>>> # Python 2.6:
>>> x = 1.
>>> while !math.isinf(x):
...   x *= 2.
...   print x
```

String literals

```
>>> print "spam"
>>> print 'spam'  # Equivalent
>>> print "spam'  # Must not mix ' and "
>>> print 'I say "spam!"' # but may nest

>>> c = "x" # Characters: strings of length 1

>>> s = """This is a
... block string
... """
>>> print s
```

Basic operations

```
>>> s = "Spam for the world!" # Assignment
>>> len(s)   # Length; no trailing NULL

>>> "spam" + "spam"  # Concatination
>>> 'spam' + "spam"


>>> 'spam' + 100 # No automatic conversion
>>> 'spam' + '100'
>>> 'spam' * 100
>>> 'spam' * '100'
```

Escape sequences

```
>>> print "First line\nSecond line"
>>> # print and not print are different:
>>> "First line\nSecond line"
>>> print "1\t2\t3\n11\t12\t13"
>>> print 'I say "spam\'s spam!"'
>>> s = "abc\
... def"
>>> print s
```

Raw strings and Unicode strings

```
>>> "C:\new\test.txt"
>>> print "C:\new\test.txt"    # whoops
>>> print "C:\\new\\test.txt"  # ok
>>> # Alternative: Raw string
>>> print r"C:\new\test.txt"

>>> # Unicode strings code extended
>>> # character sets
>>> u"äöü"   # Do not know enough...
```

Exercises

- Experiment with strings:

```
>>> print 'spam' + "spam"
>>> print "spam' + 'spam"

>>> print "#Comment or not?"
>>> print "abc\ #Comment or not?
... def"
>>> print 'I say "spam\'s spam!"'
>>> print r'I say "spam\'s spam!"'
```

Indexing

```
>>> s = "Spam for the world!"


>>> s[0]     # First character
>>> s[1]     # Second character
>>> s[18]    # Last character
>>> s[19]    # Bounds are checked!


>>> s[-1]    # ?
```

Indexing

```
>>> s[-1]    # Last character!
>>> s[-2]    # -i -> len(s)-i
>>> s[-19]   # First character
>>> s[-20]   # Bounds are checked!
>>> # Positive indices: 0 .. len(s)-1
>>> # Negative indices: -1 .. -len(s)

>>> i = 0
>>> while i<len(s):
...   print s[i], s[-(i+1)]
...   i += 1
```

Slicing

```
>>> s = "Spam for the world!"
>>> s[2:5]   # Includes s[2], but not s[5]
>>> s[5:2]   # Empty string
>>> s[2:2]   # again empty
>>> s[5:-1]  # Excludes last character
>>> s[-100:100] # Bounds are ignored here
>>> s[:5]   # Default first index: 0
>>> s[5:]   # Default last index: len(s)
>>> s[:]    # Copy of complete string
>>> s[2:10:2]
>>> s[::-1]
```

String operations

```
>>> s = "Spam for the world!"

>>> s[5] = "F"   # Strings are immutable!
>>> s = s[:5] + "F" + s[6:]

>>> s.replace("world", "universe")
>>> print s      # s is unchanged!
>>> s = s.replace("world", "universe")
```

Dot notation

```
>>> s = s.replace("F", "f")
>>> # Applies method replace to string
>>> # object s with arguments ("F", "f")
>>> x = 12
>>> x = x.replace(1, 2)
>>> # Fails, because integer objects have
>>> # no method replace
>>> z = 1+2j
>>> z.real
>>> # Returns attribute real of complex
>>> # object z
```

Some string methods

```
>>> s = "Spam for the world!"

>>> s.find("world")
>>> s.index("world")   # Identical, but:
>>> s.find("Bonn")     # -1
>>> s.index("Bonn")    # Error
```

Some string methods

```
>>> "  Hello world    ".strip()
>>> s = "Spam for the world!"
>>> s.strip("!ampS")

>>> s.split()     # List of strings
>>> s = "Spam\tfor\nthe world!"
>>> s.split()
>>> s.split("o")
```

Exercise: String methods

- Explore further methods by looking at the online documentation and trying things out

```
>>> # Hint:
>>> s = ""
>>> s.<TAB>     # Gives list of methods
>>> print s.replace.__doc__ # Prints doc
```

Converting strings and numbers

```
>>> # Numbers to strings:
>>> str(123), str(2**1000)
>>> str(1.e10), str(1.+2j)

>>> # Strings to numbers:
>>> int("123"), int("1234567890"*100)
>>> float("1.23"), float("1.23e10")
>>> float("1.23 e10")  # Error

>>> "123".isdigit()
>>> "1.23".isdigit()  # :-(
```

String formatting

```
>>> # Very similar to sprintf in C:
>>> "Spam for %s!" % "the world"
>>> "%s for %s!" % ("Ham", "us")

>>> "~%d~" % 123   # ~ for illustration
>>> "~%d~" % 1.23
>>> "~%6d~" % 123  # 6 digits (incl. sign)
>>> "~%6d~" % -12345678  # ...or more
>>> "~%-6d~" % 123  # left jusified
>>> "~%06d~" % 123  # zero padding
>>> "~%+6d~" % 123  # + displayed
```

String formatting

```
>>> "~%f~" % 1.2345
>>> # Three digits after digital point:
>>> "~%.3f~" % 1.2345
>>> # At least 8 characters
>>> # (incl. sign and decimal point)
>>> "~%8.3f~" % 1.2345
>>> "~%+08.3f~" % 1.2345

>>> "~%10.3e~" % 1.2345   # With exponent
>>> "~%10.3g~" % 1.2345   # bug?
>>> "~%10.3g~" % 12345.
```

String formatting

```
>>> # String formatting can be combined:
>>> import math
>>> s = "%s is %10.3g" % ("Pi", math.pi)
>>> print s
```

Exercises: String formatting

- Produce a pretty logarithmic table for the numbers 0.1, 0.2,... 10.0 which gives the logarithms for bases 2, e, and 10 to 7 digits.
- Hint: math.log lets you specify the base. (Consult math.log.__doc__.) Use "%10.7f" for string formatting.