Energy Informatics

Data Modeling and Analysis

Albert-Ludwigs-Universität Freiburg

UNI FREIBURG

Peter Thiemann

13 Feb 2017

Who am I?

Who are you?

Data Modeling and Analysis

- Python basics
- Python for data analytics

Jumping into Python

From the python.org website

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

Jumping into Python

From the python.org website

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

What does that mean?

Jumping into Python

From the python.org website

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

What does that mean?

- **interpreted**: interactive like a pocket calculator

Jumping into Python

From the python.org website

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

What does that mean?

- **interpreted**: interactive like a pocket calculator
- **dynamic typing**: programs just run . . .

Python as a calculator

Numbers: int, float

Syntactic elements

- int(egers): `0`, `1`, `-1`, `42`, `-32768`, ...
- float(ing point numbers): `1.0`, `3.14159`, `.2288`, `-43.4` ...
- usual arithmetic operators: `+`, `-`, `*`, `/`, `%` (remainder)

Python as a calculator

Numbers: int, float

Syntactic elements

- int(egers): `0, 1, -1, 42, -32768, ...`
- float(ing point numbers): `1.0, 3.14159, .2288, -43.4 ...`
- usual arithmetic operators: +, −, ∗, /, % (remainder)

Talking to Python

```
>>> 2 + 2
4
>>> 50 - 5*6
20
>>> (50 - 5.0*6) / 4
5.0
>>> 8 / 5.0
1.6
```

Syntactic elements

- `"a string"`
- `'Monty Python\'s flying circus'`
- Operations: concatenation, indexing, and many more

Syntactic elements

- `"a string"`
- `'Monty Python\'s flying circus'`
- Operations: concatenation, indexing, and many more

Talking to Python

```
>>> 'Monty Python\'s flying circus'
"Monty Python's flying circus"
>>> 'Monty '  'Python'          # concatenation
'Monty Python'
>>> 'Monty' + ' ' + 'Python' # concatenation
'Monty Python'
>>> 'Monty Python'[4]           # index starts at 0
'y'
```

Syntactic elements

- variable names: `x`, `y`, `tissue`, `one_of`, ...
- assignment: `x = 1`, `y = 43.2`, `tissue = 'tempo'`

Python as a calculator
Variables

Syntactic elements

- variable names: `x`, `y`, `tissue`, `one_of`, ...
- assignment: `x = 1`, `y = 43.2`, `tissue = 'tempo'`

Talking to Python

```
>>> width = 42
>>> width
42
>>> width * 2
84
>>> height
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'height' is not defined
```

Syntactic elements

- empty list: `[]`
- enumerated lists:
  `[1, 3, 5, 7, 9]`, `['a', 'e', 'i', 'o', 'u']`
- operations: index and concatenation (like string)

Syntactic elements

- empty list: `[]`
- enumerated lists:
  `[1, 3, 5, 7, 9]`, `['a', 'e', 'i', 'o', 'u']`
- operations: index and concatenation (like string)

Talking to Python

```
>>> primes = [2, 3, 5, 7, 11]
>>> primes
[2, 3, 5, 7, 11]
>>> primes[3]
7
>>> primes + [13, 17, 19]
[2, 3, 5, 7, 11, 13, 17, 19]
```

Double the input

```
>>> def double(n):   # define function named 'double'
...     return 2*n   # return value of expression
...
>>> double(21)
42
>>> double("la")     # oops
'lala'
```

Important: Indentation (PEP-8)

The function body needs to be indented by four spaces.

Pass or fail?

Evaluating a test

You can obtain a certain maximum number of marks in a test and you need at least a certain percentage of marks to pass. Write a function that takes the maximum marks, minimum percentage to pass, and the actually reached marks and returns **pass** or **fail** (as a string).

Pass or fail?

Evaluating a test

You can obtain a certain maximum number of marks in a test and you need at least a certain percentage of marks to pass. Write a function that takes the maximum marks, minimum percentage to pass, and the actually reached marks and returns **pass** or **fail** (as a string).

Comparison and Conditional

Solving this task requires a *comparison* and a *conditional*.

Comparison

Comparison Operators

- `==`, `!=` "equals" and "not equals"
- <, > "less than" and "greater than"
- <=, >= "less than or equal" and "greater than or equal"

Properties

- both operands must have the same type
- most types are sensible (numbers, strings, ...)
- result is `False` or `True`

Conditional
Examples

```
>>> if 4<5:
...     print("yes")
... else:
...     print("no")
...
yes
>>> if "max" < "fred":
...     print("max␣goes␣first")
... else:
...     print("fred␣goes␣first")
...
fred goes first
```

Pass or fail?

Python implementation

```
>>> def check_test(max_marks, percentage, marks):
...     if marks >= max_marks * percentage / 100:
...         return "pass"
...     else:
...         return "fail"
...
>>> check_test(100, 50, 49)
'fail'
>>> check_test(100, 50, 50)
'pass'
>>> check_test(100, 50, 99)
'pass'
```

Pass or fail?

Extension

What if someone calls the function with nonsense? We want the function to return the string `"illegal"` in such cases.

Pass or fail?

Extension

What if someone calls the function with nonsense? We want the function to return the string `"illegal"` in such cases.

Partial solution

```
def check_test(max_marks, p, marks):
    if p < 0 or p > 100:
        return "illegal"
    if max_marks <= 0:
        return "illegal"
    if marks < 0 or marks > max_marks:
        return "illegal"
    # rest as before
```

Pass or fail?

Extension

What if someone calls the function with nonsense? We want the function to return the string `"illegal"` in such cases.

Partial solution

```
def check_test(max_marks, p, marks):
    if p < 0 or p > 100:
        return "illegal"
    if max_marks <= 0:
        return "illegal"
    if marks < 0 or marks > max_marks:
        return "illegal"
    # rest as before
```

Python logical operators

or, **and**, **not**.

Temperature

UNI FREIBURG

Gauging the temperature of a drink

We want to gauge the temperature of (hot) coffee. The optimal drinking temperature is between 50 and 60 degrees centigrade.

Temperature

Gauging the temperature of a drink

We want to gauge the temperature of (hot) coffee. The optimal drinking temperature is between 50 and 60 degrees centigrade.

Python implementation

```
>>> def coffee_drinkable(temp):
...     return 50 <= temp <= 60
...     # returns a boolean, True or False
...
>>> coffee_drinkable(10)
False
>>> coffee_drinkable(100)
False
>>> coffee_drinkable(55)
True
```

More discerning temperature check

Coffee temperature

Given the temperature in a cup of coffee, return "too hot" if the temperature exceeds 60 degrees, "just right" if the temperature is between 50 and 60 degrees, and "too cold" if it is below 50.

Conditional for coffee judgment

```
>>> def coffee_judgment(temp):
...     if temp < 50:
...         return "too cold"
...     if temp < 60:
...         return "just right"
...     else:
...         return "too hot"
...
>>> coffee_judgment(45)
'too cold'
>>> coffee_judgment(55)
'just right'
>>> coffee_judgment(65)
'too hot'
```

Task: solve $ax^2 + bx + c = 0$ using the quadratic formula

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Implementation of quadratic formula

```
>>> import math
>>> def midnight(a, b, c):
...     return (-b + math.sqrt(b*b - 4*a*c))/2/a
...
>>> midnight(1,0,-1)
1.0
```

Looks good! 1.0 is a root of $x^2 - 1 = (x + 1)(x - 1)$

- but what about the other root -1.0 of $x^2 - 1$?

- but what about the other root -1.0 of $x^2 - 1$?
- we could return a list of roots!

- but what about the other root -1.0 of $x^2 - 1$?
- we could return a list of roots!

Revised implementation of quadratic formula

```
>>> def midnight2(a, b, c):
...     d = b*b - 4*a*c
...     return [(-b + math.sqrt(d))/2/a,
...             (-b - math.sqrt(d))/2/a]
...
>>> midnight2(1,0,-1)
[1.0, -1.0]
```

UNI FREIBURG

- but what about the other root -1.0 of $x^2 - 1$?
- we could return a list of roots!

Revised implementation of quadratic formula

```
>>> def midnight2(a, b, c):
...     d = b*b - 4*a*c
...     return [(-b + math.sqrt(d))/2/a,
...             (-b - math.sqrt(d))/2/a]
...
>>> midnight2(1,0,-1)
[1.0, -1.0]
```

- Ok, got both now . . . are we done?

More tests

Test #1

$$x^2 + 2x + 1 = 0$$

More tests

Test #1

$$x^2 + 2x + 1 = 0$$

Testing the implementation

```
>>> midnight2(1,2,1)
[-1.0, -1.0]
>>> # unsatisfactory. should return one value
```

More tests
Improving the implementation

Test #1

$$x^2 + 1 = 0$$

More tests

Improving the implementation

Test #1

$$x^2 + 1 = 0$$

Testing the implementation

```
>>> midnight2(1,0,1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in midnight2
ValueError: math domain error
>>> # oops! this equation has no real roots!
```

Facts from mathematics

Consider equation E:

$$ax^2 + bx + c = 0$$

Let $d = b^2 - 4ac$

- E has two distinct real solutions if $d > 0$
- E has one real solution if $d = 0$
- E has no real solutions if $d < 0$

We need to model this case distinction in the `midnight` function using a conditional `if`, `else`.

Case distinction: if-else

Final implementation of quadratic formula

```
>>> def midnight3(a, b, c):
...     d = b*b - 4*a*c
...     if d < 0:
...         return []
...     elif d == 0:
...         return [-b/2/a]
...     else:
...         return [(-b + math.sqrt(d))/2/a,
...                 (-b - math.sqrt(d))/2/a]
...
>>> midnight3(1,0,-1)
[1.0, -1.0]
>>> midnight3(1,2,1)
[-1]
>>> midnight3(1,0,1)
[]
```

Final thoughts

1. The present way of dealing with $d < 0$ is unsatisfactory. Python also supports complex numbers: just `import cmath` and use `cmath.sqrt` to compute the two roots in this case.
2. Try `midnight3 (0,1,2)`. What happens?

On if, elif, and else

else marks alternative block to exec

```
def f(a, b):
    d = 0
    if a > 10:
        d = 1
    else:
        d = 2
    return d
```

On if, elif, and else

`else` marks alternative block to exec

```
def f(a, b):
    d = 0
    if a > 10:
        d = 1
    else:
        d = 2
    return d
```

Example calls

- `f (0,0)` : returns 2
- `f (20,0)` : returns 1
- `f (20,20)` : returns 1
- `f (0, 20)` : return 2

On if, elif, and else II

if continues execution after indented block

```
def f(a, b):
    d = 0
    if a > 10:
        d = 1
    if b > 10:
        d = 2
    return d
```

On if, elif, and else II

`if` continues execution after indented block

```
def f(a, b):
    d = 0
    if a > 10:
        d = 1
    if b > 10:
        d = 2
    return d
```

Example calls

- `f (0,0)` : returns 0
- `f (20,0)` : returns 1
- `f (20,20)` : returns 2 (second assignment overwrites first)
- `f (0, 20)` : return 2

On if, elif, and else III

`elif` skips execution after indented block

```
def g(a, b):
    d = 0
    if a > 10:
        d = 1
    elif b > 10:
        d = 2
    return d
```

On if, elif, and else III

elif skips execution after indented block

```
def g(a, b):
    d = 0
    if a > 10:
        d = 1
    elif b > 10:
        d = 2
    return d
```

Example calls

- `g (0,0)` : returns 0
- `g (20,0)` : returns 1
- `g (20,20)` : returns 1 (`elif` skips when `a > 10`)
- `g (0,20)` : return 2

Functions
Check first letter

Task

Write a function `check_first` that takes a string and a character and checks whether it matches the first character of the string.

Functions
Check first letter

Task

Write a function `check_first` that takes a string and a character and checks whether it matches the first character of the string.

Solution

```
>>> def check_first(str, ch):
...     return str[0] == ch
...
>>> check_first('Larynx', 'L')
True
>>> check_first('atama', 'x')
False
>>> check_first([2,3,5], 2) # works for lists!
True
```

Output and formatting

Printing

Printing

The `print` statement takes any object and prints it.

Talking to Python

```
>>> print(42)
42
>>> print(4/5)
0.8
>>> print(True)
True
>>> print("flame")
flame
>>> "flame" # an expression with string value
'flame'
>>> 1*print('c')
TypeError: unsupported operand type(s) for *: 'int' and
```

Printing any object

```
>>> print([1,2,3])
[1, 2, 3]
>>> print([[], [1], [1,2]])
[[], [1], [1, 2]]
>>> def double(x):
...     return 2*x
...
>>> print(double)
<function double at 0x10acd62a8>
```

Formatted Printing

String operations create a string in the desired form, then **print**.

f-Strings

String literals with holes for expressions.

```
>>> captain = "Jim"
>>> message = f"He's␣dead␣{captain}."
"He's␣dead␣Jim."
```

Additional formatting for numbers

```
>>> f"pi␣is␣{math.pi}"
'pi␣is␣3.141592653589793'
>>> f"pi␣is␣{math.pi:10.2}"
'pi␣is␣␣␣␣␣␣␣␣␣3.1'
>>> f"pi␣is␣{math.pi:10.8}"
'pi␣is␣␣3.1415927'
```

More Formatting

String Alignment

These methods align their receiver string inside a given space of *n* characters.

- `ljust()` align to left
- `rjust()` align to right
- `center()` align in center

```
>>> "x".rjust(5)
'␣␣␣␣x'
>>> "x".ljust(5)
'x␣␣␣␣'
>>> "x".center(5)
'␣␣x␣␣'
```

More useful string operations may be found here `https://docs.python.org/3.6/library/stdtypes.html#string-methods`

String Formatter

Given a template string, the `format` method can fill in the holes.

```
>>> "Dear␣{},␣I'm␣so␣{}␣today!".format("diary", "cold")
"Dear␣diary,␣I'm␣so␣cold␣today!"
```

If the sequence of arguments is different, then the template can use explicit positions.

```
>>> "Dear␣{1},␣I'm␣{0}␣today!".format(24, "Mum")
"Dear␣Mum,␣I'm␣24␣today!"
```

Many further options to format numbers and other data.
https://docs.python.org/3.6/library/string.html#formatstrings

Input

Reading text from the console

Line Input

The console is called *standard input* `stdin`. The function `input` reads a line and returns it as a string.

```
>>> x = input()
wurstbrot
>>> x
'wurstbrot'
```

Word Input

To read multiple words, we need to split the line.

```
>>> y = input()
first things     first
>>> y
'first␣things␣␣␣␣␣first'
>>> y.split()
['first', 'things', 'first']
```

Reading numbers

Number input

Each numeric type comes with a function to read a number from a string or to convert it from another numeric format.

- `int` builds a machine integer from a string or another number
- `float` builds a floating point number . . .
- `long` builds an integer with unlimited precision
- `complex` builds a complex number

Example Uses

```
>>> int("32")
32
>>> int("-768")
-768
>>> int("␣␣␣1024␣␣␣")
1024
>>> int("␣␣␣1024␣!␣␣")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with
  base 10: '␣␣␣1024␣!␣␣'
```

Number Format

Only legal number characters allowed!

Example: Reading Input for a Task

Task

Read two integers a and b in this order from STDIN and print three lines where:

1. The first line contains the sum of the two numbers.
2. The second line contains the difference $(a - b)$.
3. The third line contains the product of the two numbers.

Input format

The first line contains the first integer, a. The second line contains the second integer, b.

Constraints

$$1 \leq a \leq 10^{10}$$

$$1 \leq b \leq 10^{10}$$

Solution

```
def solution():
    a = int(input())
    b = int(input())
    print(a+b)
    print(a-b)
    print(a*b)
```

Testing

```
>>> solution()
123
234
357
-111
28782
```

Same Task, Different Input Format

Input format

The first line contains both integers, *a* and *b*, in this order.

```
def solution2():
    line = input().split()
    a = int(line[0])
    b = int(line[1])
    print (a+b)
    print (a-b)
    print (a*b)
```

Continued

Testing

```
>>> solution2()
123 234
357
-111
28782
```

Your Task

- Can you write a solution that is flexible as to whether the input is on one or two lines?

End Part I