Basic_start

BB1000 Programming in Python KTH

Acquaintance

```
>>> print('Hi Obelix')
Hi Obelix
```

Variables: `x,y,z`

Operators: `+,-,*,/,%,**`

Expressions: `a+3, (a+b)/c`

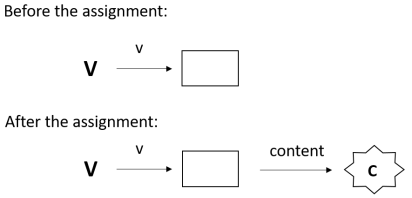
! Pay attention at the version of Python:

Assume `a=3, b=2`, then `a/b` --> `1.5` (Python3), but `1` (Python 2.7).

For Python 2.7: `float(a)/b` --> `1.5`.

[In this course, we use Phyton3]

Assignment



`=` is not the boolean expression `==` ('is equal to'), it has to be read as 'becomes':

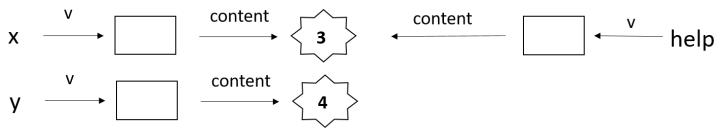
```
>>> x=3
>>> y=4
>>> x=x-y
>>> y=y-x
>>> print(x)
-1
>>> print(y)
1
```

Assignment

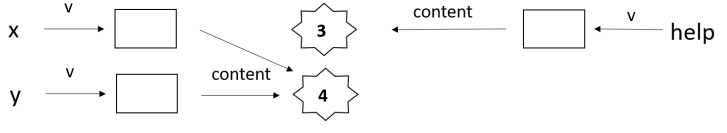
The change of the content of two variables has to be performed using a help-variable:

```
>>> help=x
>>> x=y
>>> y=help
```

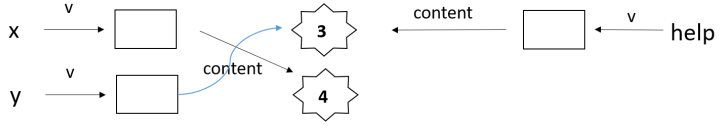
After the assignment ‘help=x’ :



After the assignment ‘x=y’ :



After the assignment ‘y=help’ :



Condition and if-statement

The syntax of a condition is `<expression><relation><expression>`, with `<relation>` denoting `==,>,<,>=,<=,!=,in`. In the last case, `something in somethingelse` investigates whether something is a member of somethingelse. This is useful e.g. to ask whether a number falls into a range of numbers.

For Python, the if-statement has a syntax as

```
>>> if <condition>:
        <statement>
>>> else:
        <statement>
```

Condition and if-statement

! Pay attention to the indentation... In Python, the absence of indentation denotes the end of the if-statement!

The case

```
>>> if <idefix is faster than asterix>:
        <idefix gets a bone>
        <Obelix is happy>
```

tells something else than

```
>>> if <idefix is faster than asterix>:
        <idefix gets a bone>
>>> <Obelix is happy>
```

Condition and if-statement

When `else` gives access to a condition, too, use can be made of `elif`:

```
>>> if <condition1>:
      <statements>
>>> elif <condition2>:
      <statements>
>>> elif <condition3>:
      <statements>
>>> else:
      <statements>
```

--> Example: how to put two numbers in ascending order

```
>>> if x > y:
      help = x
      x = y
      y = help
>>> print(x,y)
```

Repetition

Do something as long as a condition is fullfilled:

```
>>> while <condition>:
      <statements>
```

It boils down to the following algorithm:

1. Evaluate condition
2. If the condition is true, execute the statements and go back to step 1.
3. If the condition is false, stop.

--> Example: the algorithm of Euclides to find the highest common denominator...

```
>>> x=12
>>> y=15
>>> while x != y:
      if x > y:
        x = x-y
      else:
        y = y-x
>>> print('The highest common denominator is', x)
The highest common denominator is 3
```

Lists

Simple types of variables (like e.g. characters, real and integer numbers,...) can be collected in structured types. A first structured type is a list, which is given in brackets []. Be warned that the index of the list runs from 0 to the length of the list minus one!

```
>>> list = [5, 3, 'p', 9, 'e']
>>> list[0]
5
>>> list[len(list)-1]
'e'
```

`+` ('concatenation') unifies two lists:

```
>>> list2 = [2,6]
>>> list+list2
[5, 3, 'p', 9, 'e', 2, 6]
>>> len(list+list2)
7
```

Slicing of lists

```
>>> list = [5, 3, 'p', 9, 'e']
>>> list[1:3]
[3, 'p']
>>> list[2:]
['p', 9, 'e']
```

An index with a negative sign denotes counting from the end of the list to the beginning:

```
>>> list[2:-1]
['p', 9]
>>> list[2:-2]
['p']
```

List manipulation

Append, remove, insert, is a member of:

```
>>> list.append(37)
[5, 3, 'p', 9, 'e', 37]
>>> list.insert(2, 'z')
[5, 3, 'z', 'p', 9, 'e', 37]
>>> list.remove('e')
[5, 3, 'z', 'p', 9, 37]
>>> 'p' in list
True
```

A special notification for `range()`, which gives access to a list:

```
>>> range(5)[0]
0
>>> range(5)[4]
4
>>> len(range(5))
5
```

for-loop

```
>>> i=0
>>> while i in range(5):
        print(i)
        i=i+1
```

is equivalent to

```
>>> for i in range(5):
         print(i)
```

In both cases, the output is

```
0
1
2
3
4
```

Compact notation

How to make lists? Long version:

```
>>> result=[]
>>> for val in collection:
      if <condition>:
         result.append(<expression>)
```

Short version:

```
>>> [<expression> for val in collection if <condition>]
```

For example:

```
>>> [x*5 for x in range(5)]
[0, 5, 10, 15, 20]
>>> [x*5 for x in range(5) if x%2 == 0]
[0, 10, 20]
```

Strings

```
>>> string_ex='Obelix is a Celt'
'Obelix is a Celt'
>>> string_ex2 = string_ex.replace('Celt', 'strong Celt')
'Obelix is a strong Celt'
```

E.g. a number can be converted into a string using `str`.

```
>>> a=5.6
>>> s=str(a)
>>> s
'5.6'
```

Slicing is also possible with a string:

```
>>> s='menhir'
>>> s[:3]
'men'
```

Dictionary

A `dict` is built up through a collection of key-value pairs within curly braces {}. The syntax is therefore

```
>>> dict={key1: value1, key2: value2,...}
```

The breakfast of Obelix can furtheron be manipulated as

```
>>> Bf_Obelix={'egg': 5, 'wporc': 3, 'chicken': 2, 'mouse': 1}
>>> Bf_Obelix['egg']
5
>>> Bf_Obelix.keys()
dict_keys(['chicken', 'wporc', 'mouse', 'egg'])
>>> Bf_Obelix.items()
dict_items([('chicken', 2), ('wporc', 3), ('mouse', 1), ('egg', 5)])
>>> 'wporc' in Bf_Obelix
True
```

Dictionary

To delete an item from the dictionary, use

```
>>> del Bf_Obelix['mouse']
>>> Bf_Obelix.items()
dict_items([('chicken', 2), ('wporc', 3), ('egg', 5)])
```

To add an item to the dictionary, use `update`:

```
>>> Bf_Obelix.update({'roman': 2})
>>> Bf_Obelix.items()
dict_items([('chicken', 2), ('roman', 2), ('wporc', 3), ('egg', 5)])
```

Remark the random order of the items when the items of the Dict are requested. It is therefore important to be cautious about interpreting the `values` of the dict:

```
>>> Bf_Obelix.values()
dict_values([2, 5, 2, 3])
```

Tuples

A tuple is a one-dimensional, fixed-lenth immutable sequence of Python objects.

```
>>> tup = 4, 5, 6
(4, 5, 6)
>>> nested_tup = (4, 5, 6), (7, 8)
((4, 5, 6), (7, 8))
```

Any sequence can be converted to a tuple by invoking `tuple`:

```
>>> tuple([4, 5, 6])
(4, 5, 6)
>>> tuple('string')
('s', 't', 'r', 'i', 'n', 'g')
>>> tuple('string')[4]
'n'
```

Tuples

Once created, it is not possible to modify which object is stored in each slot.

```
>>> Obelix= tuple(['not thin', [1, 2], 'nectar'])
>>> Obelix[2]='magic drank'
TypeError: 'tuple' object does not support item assignment
```

From the other side, other manipulations are permitted:

```
>>> Obelix[1].append(3)
('not thin', [1, 2, 3], 'nectar')
>>> (4, None, 'stone') + Obelix
(4, None, 'stone', 'not thin', [1, 2, 3], 'nectar')
>>> (4, None, 'stone')*3
(4, None, 'stone', 4, None, 'stone', 4, None, 'stone')
```

It is straight forward to use instance methods:

```
>>> a=(4, None, 'stone')*3
>>> a.count('stone')
3
>>> 4 in a
True
```

References

"Python for Data Analysis", Wes McKinney, O'Reilly Media, Sebastopol, CA: 2013

https://perso.limsi.fr/pointal/python:memento