Programming and Data Structures with Python

Lecture 3, 21 December 2020

Names and values

Numbers

- int, float
- arithmetic operations preserve type, except / always produces float

In [1]:

```
6/3
```

In [2]:

```
(7//3,7%3)
```

In [3]:

```
(8.5//4.1,8.5%4.1)
```

Other operations, exponential

In [4]:

```
2**3  # 2 raised to 3
```

In [5]:

```
2**0.5
```

In [6]:

```
0.5**0.5
```

Math functions, $\log(x)$, $\sin(x)$ etc

In [7]:

```
log(32)
```

```
---------------------------------------------------------------
-----------------
NameError                                 Traceback (most
 recent call last)
<ipython-input-7-e31c3ec56451> in <module>
----> 1 log(32)

NameError: name 'log' is not defined
```

In [8]:

```
from math import *
```

In [9]:

```
log(32)
```

Out[9]:

```
3.4657359027997265
```

In [10]:

```
log(32,2)
```

Out[10]:

```
5.0
```

In [11]:

```
pi
```

Out[11]:

```
3.141592653589793
```

In [13]:

```
cos(pi/3)
```

Out[13]:

```
0.5000000000000001
```

Control flow

How are the steps of your program executed?

Assignment statement

In [14]:

```
x = 7
```

In [15]:

```
x
```

Out[15]:

```
7
```

In [16]:

```
y = x * 8
```

In [17]:

```
y
```

Out[17]:

```
56
```

In [18]:

```
w = z + x
```

```
---------------------------------------------------------------
-----------------
NameError                                 Traceback (most
 recent call last)
<ipython-input-18-82460d4e0d92> in <module>
----> 1 w = z + x

NameError: name 'z' is not defined
```

In [19]:

```
x = x + 1
```

In [20]:

```
x
```

Out[20]:

8

Data types --- define what operations are allowed on a value

Python --- names do not have types, only values have types

- No "declarations" in Python
- For in C/C++/Java: **int x;**, **float b;**

In Python, the type of a name is the type of its current value

Dynamic typing

In [23]:

```
y = 10
```

In [24]:

```
type(y)
```

Out[24]:

```
int
```

In [25]:

```
y = y / 1
```

In [26]:

```
type(y)
```

Out[26]:

```
float
```

Basic control flow --- sequence of assignment statements, executed one after another

Defining functions

In [27]:

```
def add3(a,b,c):  # Arguments / parameters to the function
    x = a + b + c
    return(x)
```

In [29]:

```
add3(7,8,9)
```

Out[29]:

```
24
```

As though we executed the function as follows:

```
a = 7
b = 8
c = 9
x = a + b + c
return(x)
```

In [30]:

```
def add3new(a,b,c):  # Arguments / parameters to the function
    return(a+b+c)
```

Indentation is important - be careful about tabs and spaces

In [32]:

```
def add3wrong(a,b,c):  # Arguments / parameters to the function
    x = a + b + c  # One tab
    return(x)      # Four spaces
```

In [33]:

```
def add3(a,b,c):  # Arguments / parameters to the function
    x = a + b + c
    return(x)

def add4(a,b,c,d):
    return(a+b+c+d)
```

Conditionals -- take different paths based on the values computed so far

-- Basic statement is **if**

In [34]:

```
def absvalue(x):
    if x >= 0:   # if condition:
        return(x)
    else:
        return(-x)
```

In [35]:

```
absvalue(-7)
```

Out[35]:

7

In [36]:

```
absvalue(8.5)
```

Out[36]:

```
8.5
```

Boolean values: True, False

- Operations: not - negates the value, or and and
- x or y is True if at least one of them is True
- x and y is True if both of them are True

In [37]:

```
b = True
```

In [38]:

```
b
```

Out[38]:

```
True
```

In [40]:

```
not(not(b))
```

Out[40]:

```
True
```

In [41]:

```
b or not(b)
```

Out[41]:

```
True
```

In [42]:

```
b and not(b)
```

Out[42]:

```
False
```

In [43]:

```
def inrange(a,b,x):  # Check if a <= x <= b
    if (a <= x and x <= b):
        return(True)
    else:
        return(False)
```

In [45]:

```
inrange(7,10,19)
```

Out[45]:

```
False
```

Comparisons: <, >, <=, >=

Equality: ==, !=

In [68]:

```
7 != 8
```

Out[68]:

```
True
```

assignment statement, function definition, if-else

In [48]:

```
def absvalue2(x):
    retvalue = x
    if retvalue < 0:
        retvalue = -retvalue   # No else: is needed
    return(retvalue)
```

Numbers, booleans assign individual values to names. Often need collections of values.

Simplest collection is a list - sequence of values

```
[7,8,9]    # List of values
[]         # Empty list
```

Operations on lists

- Combining two lists, concatenation [1,2,3] + [4,5,6] -> [1,2,3,4,5,6]

In [52]:

```
[1,2,3]+[4,5,6]  # Spacing inside the list is not important for us, but
```

Out[52]:

```
[1, 2, 3, 4, 5, 6]
```

In [ ]:

```
- Extract the value at a given position
- length of a list, say n values. len(l)
- Positions are numbered from 0 to n-1
- Value at position i, l[i]
```

In [53]:

```
len([1,2,3,4,5,6])
```

Out[53]:

```
6
```

In [54]:

```
l = [1,2,3]+[4,5,6]
```

In [55]:

```
len(l)
```

Out[55]:

```
6
```

In [56]:

```
l[4]   # Fifth element, l[0],l[1]...
```

Out[56]:

```
5
```

In [60]:

```
x[0]
```

```
---------------------------------------------------------------
-----------------
TypeError                                 Traceback (most
 recent call last)
<ipython-input-60-2f755f117ac9> in <module>
----> 1 x[0]

TypeError: 'int' object is not subscriptable
```

In [61]:

```
x = [10,11,12,13]
```

In [62]:

```
x[0]
```

Out[62]:

```
10
```

In [63]:

```
x / l
```

```
---------------------------------------------------------------
-----------------
TypeError                                 Traceback (most
 recent call last)
<ipython-input-63-191cab96cb37> in <module>
----> 1 x / l

TypeError: unsupported operand type(s) for /: 'list' and
 'list'
```

Python typing is dynamic (as we have seen) but strict --- illegal operations are flagged

Operating on all elements of a list:

```
for x in l:
```

Special case: list [0,1,2,..,n-1

```
range(n) -- 0,1,2,..,n-1
range(i,j) -- i, i+1, i+2, ..., j-1
```

In [72]:

```
def locate(l,x):  # Find the first position in l where x occurs
    position = -1
    for i in range(len(l)):  # i to range from 0 to len(l)-1
        if l[i] == x:
            position = i
    return(position)
```

In [ ]:

```

```

In [73]:

```
locate(x,13)
```

Out[73]:

```
3
```

What if we wanted to stop at the first position where we found x?

Different type of loop, which is governed by a condition

In [ ]:

```
def locate2(l,x):
    position = -1
    i = 0
    found = False
    while (i < len(l) and not(found)):
        if l[i] == x:
            position = i
            found = True
        i = i + 1
    return(position)
```

Names and values: numbers, booleans, lists

Control flow: assignment, function definition/return, if/else, for, while

In the gcd examples, we used **fm.append(i)** rather than **fm = fm + [i]**

In [74]:

```
l
```

Out[74]:

```
[1, 2, 3, 4, 5, 6]
```

In [75]:

```
l+7
```

```
---------------------------------------------------------------
-----------------
TypeError                                 Traceback (most
 recent call last)
<ipython-input-75-4bc2e218dcf7> in <module>
----> 1 l+7

TypeError: can only concatenate list (not "int") to list
```

In [76]:

```
l+[7]
```

Out[76]:

```
[1, 2, 3, 4, 5, 6, 7]
```

In [ ]:

```
if (locate(l,x) >= 0):
    # I did find it
else:
    # I did not find it
```