Python

Mojtaba Alaei

October 10, 2021

Content of the course

Variables

There is no predefiniton for variables in python:

```
>>> a=1
>>> type(a)
<class 'int'>
>>> b=2.0
>>> type(b)
<class 'float'>
>>> c=3/2
>>> type(c) # in python 3 division of two integer variables is float (
<class 'float'>
>>> r=1+2j
>>> type(r)
<class 'complex'>
>>> X= 2 # Python is case sensitive
>>> x= 3
>>> x+X
5
```

Print command

```
>>> print("Hello World") # in python3
Hello World
>>> print "Hello World" # in python2
  File "<stdin>", line 1
    print "Hello World" # in python2
                      ^
SyntaxError: Missing parentheses in call to 'print'
>>>
```

Integer and Float in python

Integers in Python3 can be of unlimited size:

```
>>> i=4009480932840932987429104020128402938 0
>>> print(i)
40094809328409329874291040201284029380
>>> i*i*i
64456164285638115888801902199410420241344600814069064340668461802 76
7858215353302298009165510005103491669157672000
>>>
```

The float limit in python is 16 decimal digits:

```
>>> from math import pi
>>> print(pi)
3.141592653589793
>>> # The first 31 decimal digits of pi
>>> real_pi=3.1415926535897932384626433832 79
>>> pi-real_pi
0.0
```

Integer and Float in python

The minimum and maximum float in python are 10^{-308} and 10^{308} respectively.

```
>>> a=1e10
>>> b=2.345e22
>>> a*b
2.3450000000000003e+32
>>> a=1e-308
>>> a*a
0.0
>>> a=1e308
>>> a*a
inf
```

Integer and Float in python

For integer division we should use // instead of / :

```
>>> i=12344003391939232323
>>> ii=i*i
>>> print(ii)
152374419740207273071578235032637697623 29
>>> a=ii//2 # integer division (we use // instead of /)
>>> print(a)
76187209870103636535789117516318848811 64
>>> ii-a*2
1
>>> b=ii/2 # float division
>>> print(b)
7.618720987010363e+39
>>>
```

String in python

```
>>> s="Hello World"
>>> type(s)
<class 'str'>
>>> s[1]
'e'
>>>
>>> s="Hello World"
>>> type(s)
<class 'str'>
>>> s[0]
'H'
>>> s[1]
'e'
>>> s[-1]
'd'
>>> s[-2]
'l'
>>> s[-3]
'r'
>>>
```

-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
H	e	l	l	o		W	o	r	l	d
0	1	2	3	4	5	6	7	8	9	10

Figure: Positive and negative indices of strings

String in python

```
>>> s1*2
'HelloHello'
>>> s1="Hello"
>>> s2=" World"
>>> s1+s2
'Hello World'
>>> s1*2
'HelloHello'
>>>
```

import modules in python

```
>>> from math import sin, cos, pi
>>> sin(pi)
1.2246467991473532e-16
>>>
```

```
>>> import math as m
>>> m.sin(m.pi)
1.2246467991473532e-16
>>> m.cos(m.pi)
-1.0
>>>
```

```
>>> import math
>>> math.sin(math.pi)
1.2246467991473532e-16
>>>
```

complex numbers

```
>>> r1=1+3j
>>> r2=2+4j
>>> r1.real
1.0
>>> r1.imag
3.0
>>> r1.conjugate()
(1-3j)
>>> abs(r1)
3.1622776601683795
>>> r1*r2
(-10+10j)
>>>
```

formating

`%s`	a string
`%d`	an integer
`%0xd`	an integer padded with x leading zeros
`%f`	decimal notation with six decimals
`%e`	compact scientific notation, e in the exponent
`%E`	compact scientific notation, E in the exponent
`%g`	compact decimal or scientific notation (with e)

formating, example "%6.2f"

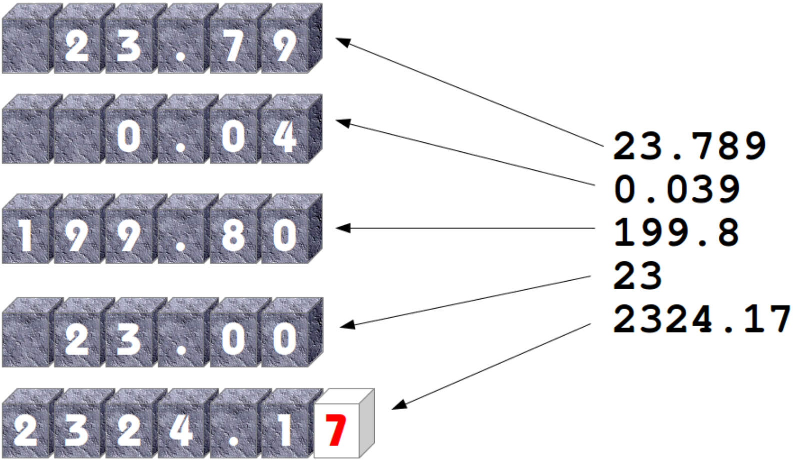

Figure: "%6.2f"

formating, example

```
>>> i=100
>>> print("%5d" % i)
  100
>>> print("%10d" % i)
       100
>>> a=1.40333e-10
>>> print("%16.5f" % a)
         0.00000
>>> print("%16.5e" % a)
     1.40333e-10
>>> print("%16.8e" % a)
  1.40333000e-10
>>> print("%20.8e" % a)
      1.40333000e-10
>>> print("%16.5E" % a)
     1.40333E-10
>>> print("%g" % a)
1.40333e-10
>>> print("%16.8g" % a)
     1.40333e-10
>>> b=0.3043245
>>> print("%16.5f" % b)
         0.30432
```

formating, example

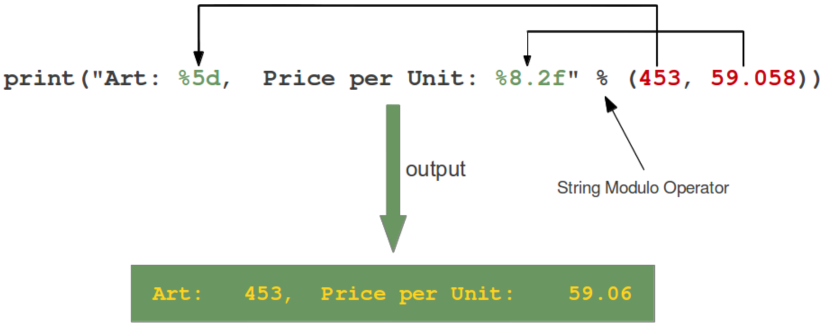

Figure:

formating, example

```python
v0 = 5
g = 9.81
t = 0.6
y = v0*t - 0.5*g*t**2
print("At t=%g s, the height of the ball is %.2f m." % (t, y))
```

```python
from math import exp, sin, pi
print("%.16f %.16f" % ( exp(-20.0), sin(pi/2-0.00001) ) )
```

formating, The string method "format"

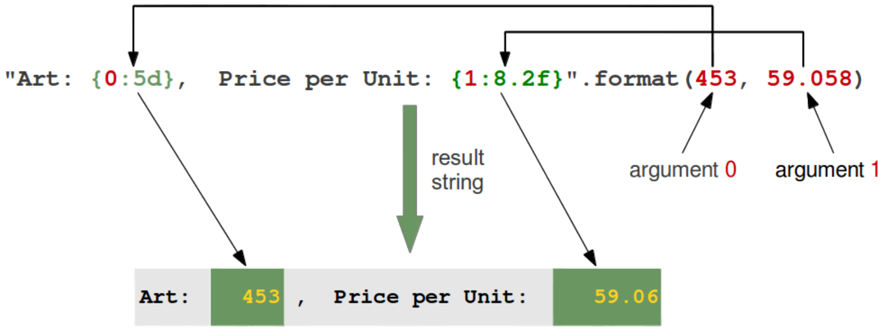

Figure:

formating, The string method "format"

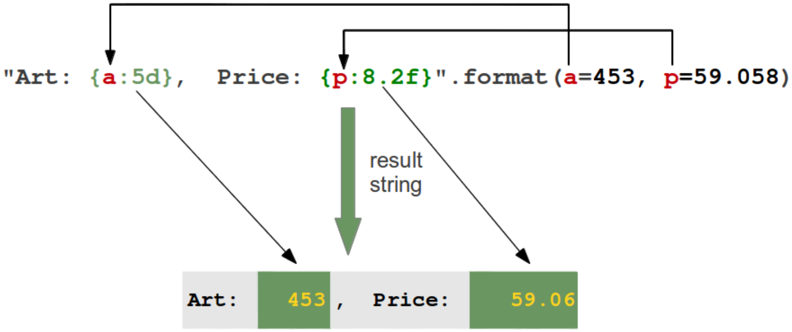

Figure:

formating, Python Print Without Newline

(this part is adapted from https://careerkarma.com/blog/python-print-without-new-line/)

```
print("First test string.")
print("Second test string.")
#output:
#First test string.
#Second test string.
```

the sloution in python 3.x is to use end:

```
print("Hello there!", end = '')
print("It is a great day.")
#output
#Hello there!It is a great day.
```

While loops

```
print("----------------")
C= -20
dC = 5
while C <= 40:
    F = (9.0/5)*C + 32
    print(C, F)
    C = C + dC
print("----------------")
```

Indentaion in Python

In Fortran to declare the body of loop or if .., we should use "END":

```
DO i=1, 100
WRITE(*,*) "i=", i
WRITE(*,*) "i=", i**2
WRITE(*,*) "i=", i**3
END DO
```

```
IF ( a > b ) THEN
WRITE(*,*) "a is larger than b"
ELSE IF
WRITE(*,*) "b is large than a or equal to a"
END IF
```

Indentaion in Python

In C and C++ to declare the body of loop or if .., we should use "{}":

```
for(i=0;i<MAXHEAP;i++){
  if(HEAP[i]==NULL){
    HEAP[i]=p;
    found=1;
    break;
  }
}
```

Indentaion in Python

In python to show body of loops or if ..., we should use indentation.

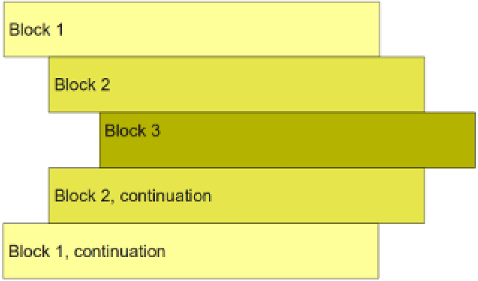

Figure: Indentation

for loop

```
>>> for i in range(0,10):
...     print(i)
...
0
1
2
3
4
5
6
7
8
9
>>>
```

for loop

```
>>> for i in range(0,10,2):
...     print(i)
...
0
2
4
6
8
>>>
```

list, examples

```
>>> C = [-10, -5, 0, 5, 10, 15, 20, 25, 30] # create list
>>> C.append(35) # add new element 35 at the end
>>> C # view list C
[-10, -5, 0, 5, 10, 15, 20, 25, 30, 35]
>>>#Two lists can be added:
>>> C = C + [40, 45] # extend C at the end
>>> C
[-10, -5, 0, 5, 10, 15, 20, 25, 30, 35, 40, 45]
>>> C.insert(0, -15) # insert new element -15 as index 0
>>> C
[-15, -10, -5, 0, 5, 10, 15, 20, 25, 30, 35, 40, 45]
>>> del C[2] # delete 3rd element
>>> C
[-15, -10, 0, 5, 10, 15, 20, 25, 30, 35, 40, 45]
>>> del C[2] # delete what is now 3rd element
>>> C
[-15, -10, 5, 10, 15, 20, 25, 30, 35, 40, 45]
>>> len(C) # length of list
11
>>> C.index(10) # find index for an element (10)
3
>>> 10 in C # is 10 an element in C?
True
>>> C.pop(1)
-10
>>> C
[-15, 5, 10, 15, 20, 25, 30, 35, 40, 45]
```

list, examples

```
>>> C[-1] # view the last list element
45
>>> C[-2] # view the next last list element
40
>>> somelist = ['book.tex', 'book.log', 'book.pdf']
>>> texfile, logfile, pdf = somelist
>>> texfile
'book.tex'
>>> logfile
'book.log'
>>> pdf
'book.pdf'
```

list, using append to creat lists

```
Cdegrees = []
n = 21
C_min = -10
C_max = 40
dC = (C_max - C_min)/(n-1) # increment in C
for i in range(0, n):
        C = C_min + i*dC
        Cdegrees.append(C)
Fdegrees = []
for C in Cdegrees:
        F = (9.0/5)*C + 32
        Fdegrees.append(F)
for i in range(len(Cdegrees)):
        C = Cdegrees[i]
        F = Fdegrees[i]
        print('%5.1f %5.1f' % (C, F))
```

list, creating a list of length n consisting of zeros

```
n = 21
C_min = -10
C_max = 40
dC = (C_max - C_min)/(n-1) # increment in C
Cdegrees = [0]*n
for i in range(len(Cdegrees)):
        Cdegrees[i] = C_min + i*dC
Fdegrees = [0]*n
for i in range(len(Cdegrees)):
        Fdegrees[i] = (9.0/5)*Cdegrees[i] + 32
for i in range(len(Cdegrees)):
        print('%5.1f %5.1f' % (Cdegrees[i], Fdegrees[i]))
```

List Comprehension

```
List = [f(i) for i in range(n)]  #list comprehension.

#f(i) represents an arbitrary mathematical operation on i

#examples

Cdegrees = [-5 + i*0.5 for i in range(n)]
Fdegrees = [(9.0/5)*C + 32 for C in Cdegrees]
C_plus_5 = [C+5 for C in Cdegrees]
```

list of list or a table as a List of Rows or Columns

```
table= [ [ 0 for i in range(6) ] for j in range(6) ]
print(table)
print("------------------------------")
for d1 in range(6):
    for d2 in range(6):
        table[d1][d2]= d1+d2+2
print(table)
print("------------------------------")
list1 = [1,2,3]
list2 = [2,3,4]
list3 = [3,4,5]
the_list = []
the_list.append(list1)
the_list.append(list2)
the_list.append(list3)
print(the_list)
print("------------------------------")
print("(0,0)", the_list[0][0])
print("(0,1)", the_list[0][1])
print("(0,2)", the_list[0][2])
```

Tuples

Tuples are very similar to lists, but (items of) tuples cannot be changed. That is, a tuple can be viewed as a "constant list". While lists employ square brackets, tuples are written with standard parentheses:

```
>>> elements = ('Cu', 'Fe', 'Co', 'O') # define a tuple with name elem
>>> t = (2, 4, 6, 'temp.pdf') # define a tuple with name t
```

One can also drop the parentheses in many occasions:

```
>>> t = 2, 4, 6, 'temp.pdf'
>>> for element in 'Cu', 'Fe', 'Co', 'O':
...     print(element)
...
Cu
Fe
Co
O
```

Tuples vs. List

Much functionality for lists is also available for tuples, for example:

```
>>> t = t + (-1.0, -2.0) # add two tuples
>>> t
(10, 11, -1.0, -2.0)
>>> t = 2, 4, 6, 'temp.pdf'
>>> t = t + (-1.0, -2.0) # add two tuples
>>>
>>> t = (2, 4, 6, 'temp.pdf')
>>> t = t + (-1.0, -2.0) # add two tuples
>>> t
(2, 4, 6, 'temp.pdf', -1.0, -2.0)
>>> t[1] # indexing
4
>>> 6 in t
True
```

Any list operation that changes the list will not work for tuples:

```
>>> t[1] = -1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> t.append(0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'append'
>>> del t[1]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object doesn't support item deletion
```

Writing over a Tuple

```
>>> t=(9,10,11)
>>> t
(9, 10, 11)
>>> t=(20,45) #Writing over a Tuple
>>> t
(20, 45)
```

Why Tuples

Some list methods, like index, are not available for tuples. So why do we need tuples when lists can do more than tuples?

- Tuples protect against accidental changes of their contents.
- Code based on tuples is faster than code based on lists.
- Tuples are frequently used in Python software that you certainly will make use of, so you need to know this data type.

Tuple with single item

Tuples are technically defined by the presence of a comma; the parentheses make them look neater and more readable. If you want to define a tuple with one element, you need to include a trailing comma:

```
>>> t=(4,)
>>> type(t)
<class 'tuple'>
>>> t1=5,
>>> type(t1)
<class 'tuple'>
```

It doesn't often make sense to build a tuple with one element, but this can happen when tuples are generated automatically.

How to Find More Python Information

```
>>> help()

Welcome to Python 3.4's help utility!

If this is your first time using Python, you should definitely check o
the tutorial on the Internet at http://docs.python.org/3.4/tutorial/.

Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules.  To quit this help utility a
return to the interpreter, just type "quit".

To get a list of available modules, keywords, symbols, or topics, type
"modules", "keywords", "symbols", or "topics".  Each module also comes
with a one-line summary of what it does; to list the modules whose nam
or summary contain a given string such as "spam", type "modules spam".

help> keywords

Here is a list of the Python keywords.  Enter any keyword to get more

False               def                 if                  raise
None                del                 import              return
True                elif                in                  try
and                 else                is                  while
as                  except              lambda              with
assert              finally             nonlocal            yield
break               for                 not
class               from                or
continue            global              pass

help>
```

How to Find More Python Information

- In the terminal type: pydoc math
- >>> import math
  >>> help(math)

How to Find More Python Information

- http://www.python-course.eu/
- https://realpython.com/
- Learn Python - Android Apps on Google Play
- http://www.tutorialspoint.com/python3/
- https://www.w3schools.com/python/
- online interactive tutorials:
  - http://www.learnpython.org/en/
  - https://www.codecademy.com/learn/learn-python-3

Functions

- Define a function

```
#define function
def F(C):
    return (9.0/5)*C + 32
#use function
Fdegrees = [F(C) for C in Cdegrees]
```

- Local and Global Variables

```
print(sum) # sum is a built-in Python function
sum = 500 # rebind the name sum to an int
print(sum) # sum is a global variable

def myfunc(n):
    sum = n + 1
    print(sum) # sum is a local variable
    return sum
sum = myfunc(2) + 1 # new value in global variable sum
print(sum)
```

Using global variables in functions

Example 1

```
a = 20; b = -2.5 # global variables
def f1(x):
    a = 21          # this is a new local variable
    return a*x + b #  21*x - 2.5
print(a) # yields 20
def f2(x):
  global a
  a = 21          # the global a is changed
  return a*x + b # 21*x - 2.5
f1(3); print(a) # 20 is printed
f2(3); print(a) # 21 is printed
```

Using global variables in functions

Example 2: Using function without return keyword!

```
>>> my_mony=10
>>> def add_mony(n):
...     global my_mony
...     my_mony=my_mony+n
...
>>> add_mony(2)
>>> my_mony
12
```

Functions

Multiple Arguments

```
def yfunc(t, v0):
    g = 9.81
    return v0*t - 0.5*g*t**2
#valid calls
y = yfunc(0.1, 6)
y = yfunc(0.1, v0=6)
y = yfunc(t=0.1, v0=6)
y = yfunc(v0=6, t=0.1)
```

Multiple Return Values

```
def yfunc(t, v0):
    g = 9.81
    y = v0*t - 0.5*g*t**2
    dydt = v0 - g*t
    return y, dydt
#the function returns two values:
position, velocity = yfunc(0.6, 3)
```

Function

- Keyword Arguments

```
from math import pi, exp, sin
def f(t, A=1, a=1, omega=2*pi):
    return A*exp(-a*t)*sin(omega*t)
#Calling f with just the t argument specified is possible:
v1 = f(0.2)
v2 = f(0.2, omega=1)
v3 = f(1, A=5, omega=pi, a=pi**2)
v4 = f(A=5, a=2, t=0.01, omega=0.1)
v5 = f(0.2, 0.5, 1, 1)
```

Function

- Doc Strings

```
def C2F(C):
    """Convert Celsius degrees (C) to Fahrenheit."""
    return (9.0/5)*C + 32
def line(x0, y0, x1, y1):
    """
    Compute the coefficients a and b in the mathematical
    expression for a straight line y = a*x + b that goes
    through two points (x0, y0) and (x1, y1).
    x0, y0: a point on the line (floats).
    x1, y1: another point on the line (floats).
    return: coefficients a, b (floats) for the line (y=a*x+b).
    """
    a = (y1 - y0)/(x1 - x0)
    b = y0 - a*x0
    return a, b
print(line.__doc__) #to see document of line function"
```

Functions as Arguments to Function

A function for computing the second-order derivative of a function f(x) numerically:

$$f'' \approx \frac{f(x-h) - 2f(x) + f(x+h)}{h^2} \quad (1)$$

```
def diff2(f, x, h=1E-6):
    r = (f(x-h) - 2*f(x) + f(x+h))/(h*h)
    return r
```

Functions as Arguments to Function

The Behaviour of the Numerical Derivative as $h \to 0$, round-off errors:

```
def g(t):
    return t**(-6)
def diff2(f, x, h=1E-6):
    r = (f(x-h) - 2*f(x) + f(x+h))/(h*h)
    return r

for k in range(1,15):
    h = 10**(-k)
    d2g = diff2(g, 1, h)
    print('h=%.0e: %.5f' % (h, d2g))
```

What will you see at the ouput?

Another example of round-off errors

```
a=0
N=100000000
for i in range(N):
    a=a+0.1
print(a-0.1*N)
```

What will you see at the ouput?

A simple example of round-off error

```
>>> 1.2+2.4  # = 3.6
3.5999999999999996
>>> 1.2+2.4-3.6 #  =  0.0
-4.440892098500626e-16
>>> 1.2+2.4 == 3.6
False
```

So how to compare float numbers?
The trick is to use a threshold for comparison:

```
>>> epsilon=1e-8
>>> abs (1.2+2.4 - 3.6 ) < epsilon
True
```

Lambda Functions

```
f = lambda x: x**2 + 4
#This so-called lambda function is equivalent to writing
def f(x):
    return x**2 + 4
#In general,
def g(arg1, arg2, arg3, ...):
    return expression
#can be written as
g = lambda arg1, arg2, arg3, ...: expression

#insert a lambda function as the f argument in the call to diff2:
d2 = diff2(lambda t: t**(-6), 1, h=1E-4)
d2 = diff2(lambda t, A=1, a=0.5: -a*2*t*A*exp(-a*t**2), 1.2)
```

Comparison Operators in Python

Operators	Meaning	Example	Result
<	Less than	5<2	False
>	Greater than	5>2	True
<=	Less than or equal to	5<=2	False
>=	Greater than or equal to	5>=2	True
==	Equal to	5==2	False
!=	Not equal to	5!=2	True

Figure: Comparison Operators

Comparison Operators in Python

```
>>> 4 < 5
True
>>> 4 < 3
False
>>> 4==4
True
>>> 4.0==4.0
True
>>> 4.0==4.000000000000000001
True
>>> 4.0==4.00000000000001
False
>>> 'Test'=='Test'
True
>>> 'Test'=='Test1'
False
>>> 7!=6
True
>>> 7!=7
False
>>> 4.0!=4.000001
True
>>> 4.0!=4.000000000000001
True
>>> 7 <= 7
True
>>> 7 >=8
False
```

Branching

$$f(x) = \begin{cases} sin(x) & 0 \leqslant x \leqslant \pi \\ 0 & \text{otherwise} \end{cases}$$

```
def f(x):
    if 0 <= x <= pi:
        value = sin(x)
    else:
        value = 0
    return value
```

If-Else Blocks

```
if condition1:
   <block of statements>
elif condition2:
   <block of statements>
elif condition3:
   <block of statements>
else:
   <block of statements>
<next statement>
```

```
def N(x):
    if 0 <= x < 1:
        return x
    elif 1 <= x < 2:
        return 2 - x
    else:
        return 0
```

inline If

```
if condition:
   a = value1
else:
   a = value2
#Python provides a inline syntax for the four lines above:
a = (value1 if condition else value2)
def f(x):
    return (sin(x) if 0 <= x <= 2*pi else 0)
#lambda functions:
f = lambda x: sin(x) if 0 <= x <= 2*pi else 0
```

Recursion function in python

```
def factorial(n):
    if n < 0 or type(n)!=int :
        return "factorial is only defined for positive integer numbers (
    elif n==0:
        return 1
    else:
        return n*factorial(n-1)
```