Statements (1A)

Young Won Lim
5/10/23

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using OpenOffice.

Variables and Types (1)

C variables are statically typed
need to specify whether a variable x is
an int or a float right up front
before using the variable

In Python, you don't:

```
x = 1
type(x)
int
```

no need to declare variables
just use them whenever we need to,
without declaring them

the type of **x** is an **int**,
x is a reference to an integer object,
which has the value 1.

https://engineering.purdue.edu/~milind/datascience/2018spring/notes/lecture-2.pdf

Variables and Types (2-1)

x itself doesn't have a fixed type.
x can be re-assigned

```
x = 1.2
type(x)
float

x = "hello"
type(x)
str
```

https://engineering.purdue.edu/~milind/datascience/2018spring/notes/lecture-2.pdf

Dynamic type checking

Python does not say
whether the operation is legal or not:
until you try to do something with a variable

```
len(x)
5
```

this will work because **x** is a string **“hello”**

```
x = 1.2
len(x)
---------------------------------------------------------------------------
TypeError Traceback (most recent call last)
<ipython-input-5-31756c9ed6f5> in <module>()
      1 x = 1.2
----> 2 len(x) #what will happen here?
TypeError: object of type 'float' has no len()
```

https://engineering.purdue.edu/~milind/datascience/2018spring/notes/lecture-2.pdf

Type coercion

type coercion

when it makes sense, it will convert an object
from one type to another to let an operation work

```
p = 1
print (type(p))

q = .2
print (type(q))

r = p + q
print (type(r))
print ("value of r: {}".format(r))
```

```
<type 'int'>
<type 'float'>
<type 'float'>
value of r: 1.2
```

https://engineering.purdue.edu/~milind/datascience/2018spring/notes/lecture-2.pdf

print statement (1)

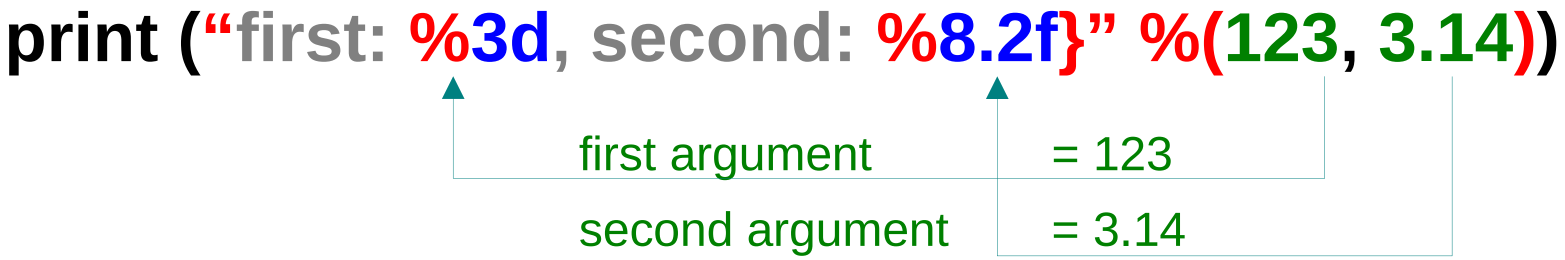
the parentheses around the argument to **print**
are optional

```
p = 1
print type(p)

q = .2
print type(q)

r = p + q
print type(r)
print "value of r: {}".format(r)
```

formatted print

https://engineering.purdue.edu/~milind/datascience/2018spring/notes/lecture-2.pdf

print statement (2)



```
print (“first: %3d, second: %8.2f}” %(123, 3.14))
```

first argument = 123

second argument = 3.14

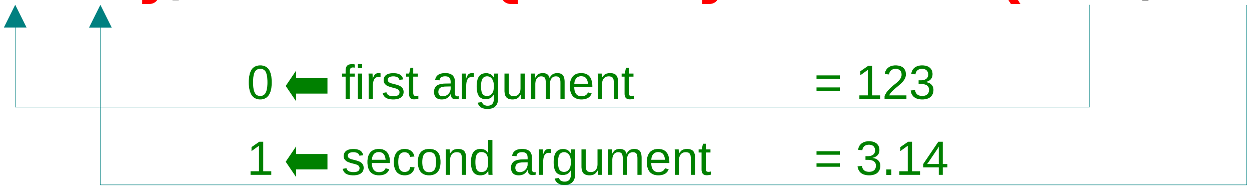
first: 123, second: 3.14

3d   **8.2f**

https://www.geeksforgeeks.org/python-output-formatting/

print statement (3)

```
print (“first: {0:3d}, second: {1:8.2f}”.format(123, 3.14))
```



0 ⬅ first argument = 123

1 ⬅ second argument = 3.14

```
first: 123, second:     3.14
```

3d **8.2f**

https://www.geeksforgeeks.org/python-output-formatting/

Control Statements (1)

Control statements in **Python**
Is similar to their counterparts in **C**:

- **if** statements,
- **while** loops,
- **for** loops.

The biggest difference is that
in **Python** whitespace matters.

Python does not use **{** and **}**
to separate blocks.

instead, **Python** use

- colons (**:**) to mark the beginning of a block and
- indentation to mark what is in the block.

https://engineering.purdue.edu/~milind/datascience/2018spring/notes/lecture-2.pdf

if statement (1)

If Statements

Here is the equivalent of the C statement:

```
if (r < 3) printf("x\n");
else printf("y\n");
```

```
if r < 3:
    print "x"
else:
    print "y"
```

https://engineering.purdue.edu/~milind/datascience/2018spring/notes/lecture-2.pdf

if statement (2)

And an example of **multiline blocks**:

```
if r < 1:
    print "x"
    print "less than 1"
elif r < 2:
    print "y"
    print "less than 2"
elif r < 3:
    print "z"
    print "less than 3"
else:
    print "w"
    print "otherwise!"
```

https://engineering.purdue.edu/~milind/datascience/2018spring/notes/lecture-2.pdf

while loop (1)

```
x = 1
y = 1
while (x <= 10) :
      y *= x
      x += 1
print y
```

https://engineering.purdue.edu/~milind/datascience/2018spring/notes/lecture-2.pdf

while loop (2)

```
x = 1
y = 1
while (x <= 10) :
    if x
```

```
File "<ipython-input-10-0f64722897ca>", line 4
if x
^
SyntaxError: invalid syntax
```

https://engineering.purdue.edu/~milind/datascience/2018spring/notes/lecture-2.pdf

for loop (1)

for loops in Python
are not like those in C

instead, iterate over **collections** (e.g., **lists**).

are more like **foreach** loops in other languages

for (x : list) construct in Java

https://engineering.purdue.edu/~milind/datascience/2018spring/notes/lecture-2.pdf

for loop (2)

```
data = [1, 4, 9, 0, 4, 2, 6, 1, 2, 8, 4, 5, 0, 7]
print data
```

```
[1, 4, 9, 0, 4, 2, 6, 1, 2, 8, 4, 5, 0, 7]
```

```
hist = 5 * [0]
print hist
```

```
[0, 0, 0, 0, 0]
```

https://engineering.purdue.edu/~milind/datascience/2018spring/notes/lecture-2.pdf

for loop (3)

Lists work like a combination of
  arrays in C
    (you can access them using **[ ]**) and
  lists
    (you can append elements, remove elements, etc.)

```
L = len(data)
print "data length: {} data[{}] = {}".format(L, L-1, data[L-1])
```

```
data length: 14 data[13] = 7
```

```
data.append(8)
L = len(data)
print "data length: {} data[{}] = {}".format(L, L-1, data[L-1])
```

```
data length: 15 data[14] = 8
```

https://engineering.purdue.edu/~milind/datascience/2018spring/notes/lecture-2.pdf

for loop (4)

You can then iterate over the elements of the list:

```
for d in data :
    print d
```

```
for d in data :
    hist[d / 2] += 1
print hist
```

```
[4, 2, 4, 2, 3]
```

```
1    0    0
4    2    0
9    4    0
0    0    0
4    2    1
2    1    1
6    3    2
1    0    2
2    1    2
8    4    2
4    2    3
5    2    3
0    0    4
7    3    4
8    4    4
```

https://engineering.purdue.edu/~milind/datascience/2018spring/notes/lecture-2.pdf

for loop (5)

write a **for** loop with an index variable
that counts from 0 to 4, like in C?

```
for (int i = 0; i < 5; i++)
```

Use the standard function range,
which generates a list with values
that count from a lower bound
to an upper bound:

```
r = range(0,5)
print r
```

```
[0, 1, 2, 3, 4]
```

```
for i in range(0, 5):
    print i
```

https://engineering.purdue.edu/~milind/datascience/2018spring/notes/lecture-2.pdf

Functions (1)

Basic functions in Python work a lot like functions in C.

The key differences are:

1. You don't have to specify a **return type**.
   In fact, you can return more than one thing!

2. You don't have to specify
   the **types** of the **arguments**

3. When calling functions,
   you can name the **arguments**
   (and thus change the **order** of the call)

```
def foo(x) :
    return x * 2

print foo(10)
```

https://engineering.purdue.edu/~milind/datascience/2018spring/notes/lecture-2.pdf

Functions (2)

```
def foo2(x) :
    return x * 2, x * 4

(a, b) = foo2(10)
print a, b
```

```
def foo3(x, y) :
    return 2 * x + y

print foo3(7, 10)
print foo3(y = 10, x = 7)
```

```
def foo2(10) :
    return 10 * 2, 10 * 4

(a, b) = foo2(10)
```

```
def foo3(7, 10) :
    return 2 * 7 + 10

print foo3(7, 10)
print foo3(y = 10, x = 7)
```

https://engineering.purdue.edu/~milind/datascience/2018spring/notes/lecture-2.pdf

Functions (3)

There are more complicated things you can do with functions -- nested functions, functions as arguments,
functions as return values, etc. We will look at these in the lecture when we talk about Map and Reduce

https://engineering.purdue.edu/~milind/datascience/2018spring/notes/lecture-2.pdf

References

[1] Essential C, Nick Parlante
[2] Efficient C Programming, Mark A. Weiss
[3] C A Reference Manual, Samuel P. Harbison & Guy L. Steele Jr.
[4] C Language Express, I. K. Chun