

Basic Data Types and Operators

CS 8: Introduction to Computer Science, Winter 2019
Lecture #2

Ziad Matni, Ph.D.
Dept. of Computer Science, UCSB

Your Instructor

Your instructor: **Ziad Matni, Ph.D** (*zee-ahd mat-knee*)

Email: ***zmatni@cs.ucsb.edu***

(please put **CS8** at the start of the subject header)

My office hours:

Wednesdays 1:00 PM – 2:30 PM, at SMSS 4409

A Word About Registration for CS8

- This class is **FULL**,
 & the waitlist is **CLOSED**.

Administrative

- Lab00 – success or not?
- Pairing in Labs
 - When doing paired labs, you may ONLY pair with someone in the SAME lab as you!
 - The TAs and I will *assign* you your lab partner, based on the ic00 form you gave us.

Linux Workshop?

- Basic commands
- How to log in remotely from your computer to CSIL
 - Copy files back and forth
- How many are interested in attending this?
- Would be on a Friday morning (10 am)
- Would last < 1 hour

Python IDLE

- *IDLE* is what we use to demonstrate Python in class
 - You can also use it at home (download info given last class)
- If you want to create a *Python program*, then you will place *all* the program code inside a text file
 - Text file always ends in **.py**
 - You can *create* and also *run (execute)* the **.py** program from Python IDLE
- Make sure the version of IDLE you use is **AT LEAST 3.7.x or LATER**

Lecture Outline

- Numbers and Arithmetic in Python
- Variables in Python
- Variable Types in Python
- Operations in Python
- Assignment versus Comparison of Values

Yellow Band = Class Demonstration! 😊

Numbers are **Objects** to Python

- Each object *type* has: **data** and related **operations**
- 2 basic number types
 - **Integers** (like **5** or **-72**) – add, subtract, multiply, ...
 - **Floating point** numbers (like **0.005** or **-7.2**) – operations similar but *not exactly the same as integer* operations
- Expect many ***non-number object*** types later in the quarter...
 - But they also have data and related operations

Common Data Types

Type	Example	Description
float	3.1415	A real number. Can be positive or negative.

Common Data Types

Type	Example	Description
float	3.1415	A real number. Can be positive or negative.
int	3	An integer number. Can be positive or negative.
string	"ILUVCS8!!!" "Gaucho Goop"	A series (or a string) of characters. Note the use of " " as delimiters.
bool	True False	A Boolean outcome of a logical comparison .

Arithmetic Operators

- $+$ $-$ $*$ $/$ add, subtract, multiply, (ordinary) divide
- $\%$ modulus operator – remainder
- $()$ means whatever is inside is evaluated first
- $**$ raise to the power

Special Python division operator for integers:

$//$ result is truncated: $7 // 2 \rightarrow 3$ (not 3.5)

What's easier to remember:

$$3 * 2 - 1$$

Or:

$$(3 * 2) - 1 ???$$

Precedence rules:

1. $()$
2. $**$
3. $*$, $/$, $\%$, $//$
4. $+$, $-$
5. $=$

Comments in Python

- Anything placed after the # symbol is considered a “comment”
 - Is completely ignored by the compiler
 - Typically place commentary next to code for the benefit of others (humans) reading our code

Variables

- A **variable** is a *symbolic* reference to data
- The variable's **name** represents *what* information it contains



- They are called “**variables**” because
 --- data can VARY or change ---
while **operations** on the variable remain the same
 - e.g. Variables “a” and “b” can take on different *values*,
 but I may always want to add them together

Variables



- Variables are like “buckets” that can keep data
 - You can label these buckets with a **name**
 - When you reference a bucket, you use its name, not the data stored in the bucket
 - You can “re-use” the buckets
- If two variables are of the same ***type***, you can perform ***operations*** on them

Variables in Python

- We assign a **value** to variables with the *assignment operator* =
 - Example: `>>> a = 3`
- We can change that value stored
 - Example: `>>> a = 5 # not 3 any more!!!`

Assigning Names to Variables

- Variable names are actually references
- Like “pointers” to objects
- Can have multiple references to the same object

`x = 5` `# x refers to an integer`

`y = x` `# Now x and y refer to the same object`

Assigning Names to Variables

- **Dynamic typing** is a key Python feature
- Any legal name can point to any ***data type*** – even different types at different times

```
x = 5          # x refers to an integer
y = x          # Now x and y refer to the same object
               # and both are integers
x = 1.2        # Now x refers to floating point 1.2
               # (y still refers to the integer 5)
```

All Data in Python Has a *type()*

- But you can change its type
 - *Implicitly*, like in the last slide
 - *Explicitly*, by forcing the type
- Introducing the built-in function *type()*
- Let's try these out on IDLE:

```
>>> grade = 3.8
>>> type( grade )
>>> grade = 5
>>> type( grade )
>>> type("Green Eggs")
>>> type(True)
>>> type(true)
```

```
>>> pi = 3.14
>>> type( pi )
>>> p = int( pi )
>>> print( p )
>>> type ( p )
```

More on “Type Casting”

- Let's try these out on IDLE and explain them:

```
>>> int(4.2)
>>> int (True)
>>> int (False)
>>> float(False)
>>> float ( true )
>>> float (4) / 5
>>> str ( 42 )
>>> int (“42”)
```

Variable Names in Python

3 simple rules for choosing names:

- Can ONLY have **letters**, **digits**, and **_** (underscores)
- Must NOT *begin with* a digit or non-alphabet character (except underscore)
- Cannot use Python reserved **keywords**
 - Example: **def**, **int**, **False**, **True**, **print**, etc...

UserName



Age1

Age2

_Deviation

2Good2BTrue



\$\$MaMoney!!

<0_0>

#YOLO

Variable Names in Python: Other Conventions

- Choose brief, but *meaningful* names
- Most programmers prefer lower case use (Example: **total** vs. **TOTAL**)
- Use either “**camel case**” or **underscore** to separate words
 - Camel Case is using capital letters to separate words, like **NumOfCats**
 - Underscoring is using underscores to separate words, like **num_of_cats**
 - **Be consistent**: use one or the other throughout your program
- All the above applies to function names, module names, etc...

Objects

- An *object* in Python is anything that has:
 - an identity a type a value
- Example: `pi = 3.14159`
 - Identity: pi
 - Type: floating point
 - Value: 3.14159

Demo

Let's try this out – what do you think it'll do?

```
pi = 3.14159  
radian_angle = 0.7853975  
degree_angle = radian_angle*180/pi  
print(degree_angle)    # What is print()?
```

Let's try it out!

The Equals Operators

= **Assign it to**

== **Is it equal to?**

!= **Is it not equal to?**

The Equals Operators

bebe = 22

The *int* variable **bebe** now has the value **22** (i.e. it is *assigned* the value 22)

bebe == 22

Is **bebe** equal to **22**? The answer is yes, in other words, the answer is Boolean **True**.

*Note that **bebe** is not changed – it's still 22*

bebe != 22

Is **bebe** not equal to 22?

The answer is **False**. Again, **bebe** is unchanged.

Let's try it out!

Assignment vs. Comparison

What happens when I do this?

	<i>a</i>	<i>b</i>
a = 1.1	1.1	
b = a	1.1	1.1
a = 2.2	2.2	1.1

Assignment vs. Comparison

What happens when I do this?

	<i>z</i>	<i>y</i>	<i>x</i>
$z = 6$	6		
$y = 8$	6	8	
$z = z + y$	14	8	
$x = (z == 14)$	14	8	True

Assignment vs. Comparison

What happens when I do this?

	<i>m</i>	<i>n</i>
m = 42	42	
n = 2 - m	42	-40
n == 0	42	-40
m = (m == 40)	False	-40

Input and Output

- We'll make use of 2 built-in functions in Python:
 - **print()** to print out to the screen
 (called *standard output*)
 - **input()** to get input from the keyboard
 (called *standard input*)

Input and Output

- To output data, use print()

```
>>> print("Hello all you happy people!")
```

- To get data and put it in a variable, use input()

```
>>> name = input()
```

OR

```
>>> name = input("Name. Give it. Now: ")
```

Let's try it out!

YOUR TO-DOs

- ☐ Read **Chapter 2**
- ☐ Finish **Homework1** (due **Monday!**)
- ☐ Prepare for **Lab1** next week

- ☐ Hug a tree! *But don't get wet...*

</LECTURE>