Fundamentals of Computer Science 30398

Lecture 2

Setting up VS Code

Extensions:

- Python
- Code Runner
- Jupyter

Setting up VS Code

Extensions:

- Python
- Code Runner
- Jupyter

Jupyter extension

Magic line

#%%

separates code into cells. We can execute cells separately in any order by pressing "Run cell" or SHIFT+RETURN.

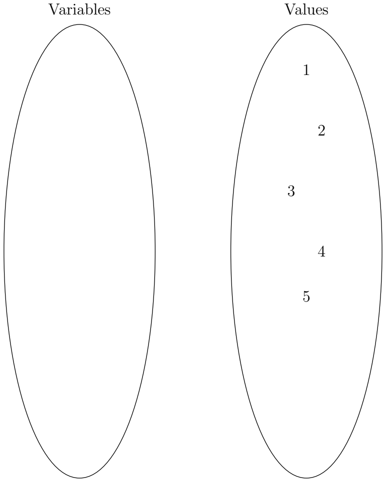
Variables, values and types

In Python, imagine that there is an abstract “universe of values”, each value has a type associated with it.

Content	Example values	Python type
Integer numbers	2, 6, 10, -11	**int**
Floating point numbers	2.13, 0.0, -1.33e10	**float**
Truth values (Boolean)	True, False	**bool**
Strings	"Hello World!", "1234"	**str**
Lists	[1,5,7], [5, "Helllo", True]	**list**

Variables, values and types

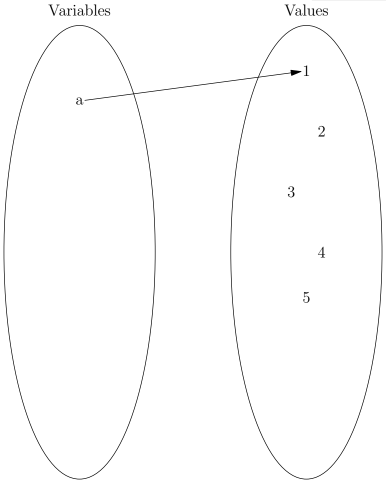
Python variables contain references to the values.



```
a = 1
a = a+1
b = a
a = a+1
```

Variables, values and types

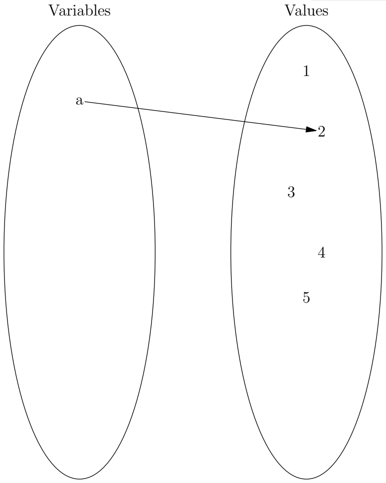
Python variables contain references to the values.



```
a = 1
a = a+1
b = a
a = a+1
```

Variables, values and types

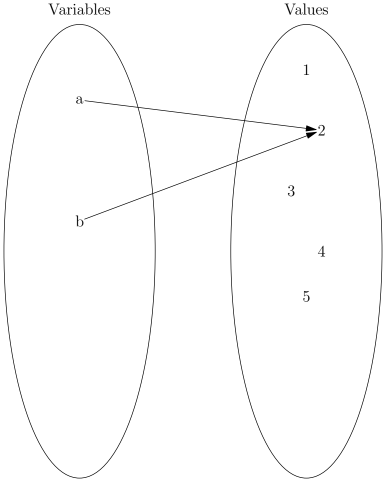
Python variables contain references to the values.



```
a = 1
a = a+1
b = a
a = a+1
```

Variables, values and types

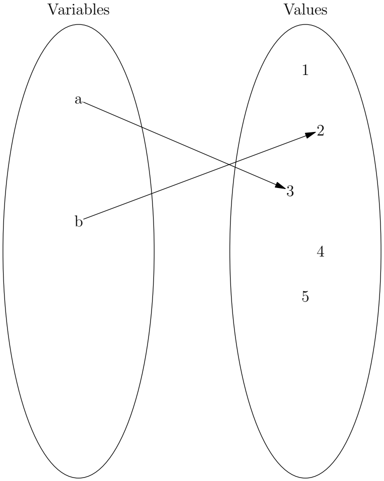
Python variables contain references to the values.



a = 1
a = a+1
b = a
a = a+1

Variables, values and types

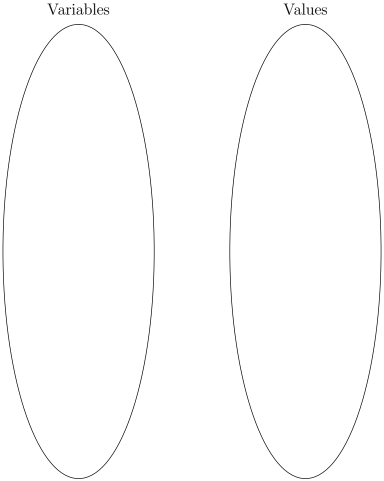
Python variables contain references to the values.



```
a = 1
a = a+1
b = a
a = a+1
```

Why this matter?

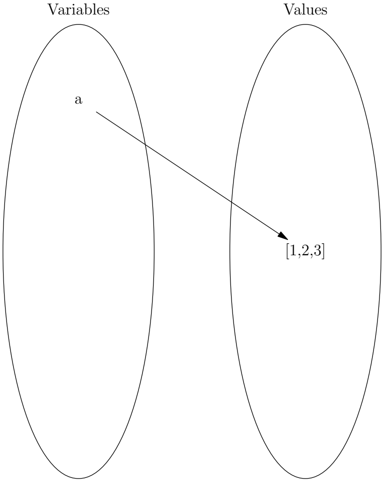
Some types are mutable. (All the remainig are immutable.) We can change a value two different variables refer to. It can be confusing at first.



```
a = [1,2,3]
b = a
a[2] = 7
```

Why this matter?

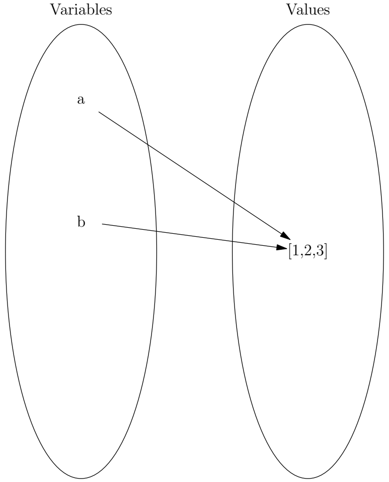
Some types are mutable. (All the remainig are immutable.) We can change a value two different variables refer to. It can be confusing at first.



```
a = [1,2,3]
b = a
a[2] = 7
```

Why this matter?

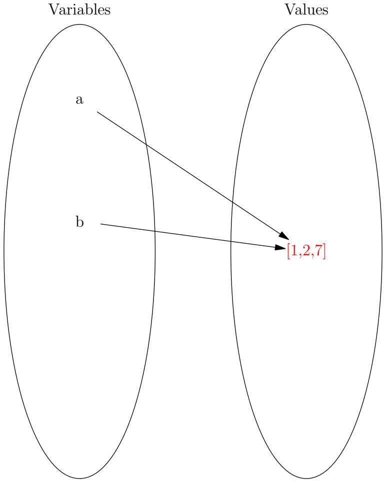
Some types are mutable. (All the remainig are immutable.) We can change a value two different variables refer to. It can be confusing at first.



```
a = [1,2,3]
b = a
a[2] = 7
```

Why this matter?

Some types are mutable. (All the remainig are immutable.) We can change a value two different variables refer to. It can be confusing at first.



```
a = [1,2,3]
b = a
a[2] = 7
```

Of the types discussed so far, only lists are immutable.

Python type	Example values	Mutable?
int	2, 6, 10, -11	
float	2.13, 0.0, -1.33e10	
bool	True, False	
str	"Hello World!", "1234"	
list	[1,5,7], [5, "Helllo", True]	Yes

Printing things

To print things in our code we use a function call **print**().
(We will discuss functions in more details later. For now —just use it.)

Example 1

```
print("Hello world")
```

Printing things

To print things in our code we use a function call **print**().
(We will discuss functions in more details later. For now —just use it.)

Example 1

```
print("Hello world")
```

Example 2

```
x = 10
print("Value of variable x is ", x)
```

(Some) integer operations

Operation	Example	Value	Value type
Addition (+)	5 + 7	12	**int**
Substraction (-)	5 - 7	-2	**int**
Multiplication (*)	5*7	35	**int**
Division (/)	5/7	0.714...	**float**
Integer division (//)	11 // 5	2	**int**
Division reminder (%)	11 % 5	1	**int**
Comparisons ($<$, $>$, $>=$, ...)	$5 < 7$	True	**bool**
Equality comparison ($==$)	$5 == 7$	False	**bool**

Control flow (Live coding)

Python code in a single cell is typically executed line-by-line from top to bottom. Unless. . .

Control flow (Live coding)

Python code in a single cell is typically executed line-by-line from top to bottom. Unless...

Conditional construction (**if**)

```
some_code
if condition:
        conditional_code
        conditional_code
other_code
```

Control flow (Live coding)

Python code in a single cell is typically executed line-by-line from top to bottom. Unless...

Conditional construction (**if**)

```
some_code
if condition:
        conditional_code
        conditional_code
other_code
```

Here the conditional code is executed only if condition is satisfied.

Example

```
n = 10
if n < 5:
        print("n is smaller than 5")
print("Rest of the code")
```

if-else construction (Live coding)

Conditional construction (**if-else**)

```
some_code
if condition:
        conditional_code_true
        conditional_code_true
else:
        conditional_code_false
        conditional_code_false
other_code
```

while loop (live coding)

```
some_code
while condition:
        looped_code
other_code
```

While loop

When Python encounters **while** instruction, it checks condition. If it is true, it executes the looped code, and goes back to check the condition. If it is false, it skips straight to other code.

while loop example (live coding)

Example

```
i = 0
while i < 10:
        print("i is", i)
        i = i + 1
print("Finished")
```