UNIVERSITY OF TWENTE.

Variables and Types in Python

Presented by:
Mahdi Farnaghi
Assistant Professor,
Department of Geo-information Processing
ITC, The University of Twente

Python as calculator

```
# Simple calculations
2 + 2
4

# Calculations with variables
x = 22
y = 45
x * y
990

# Python can handle loooong numbers too:
x = 213234734873487247124724452345456456845684 56
y = 23495095409890345093451304598734957 3485
x * y
500997044055525491568816109342578423463980851923779466
   1894041381931115877884989160
```

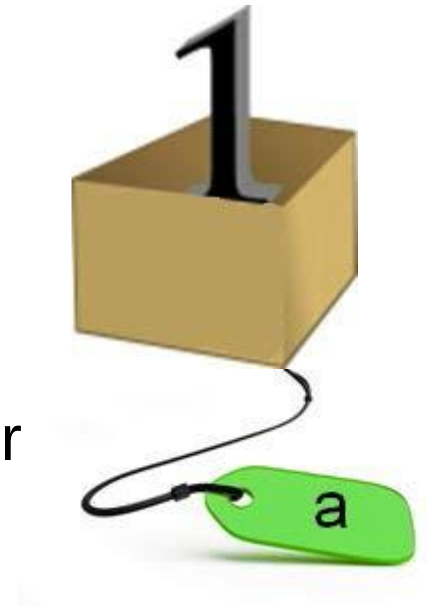
UNIVERSITY OF TWENTE.

Variables

A variable is like a labelled box that can store values

A variable always has:

an identifier (aka name)	a
a value	1
a type	integer



Assignment statement

In mathematics,

$x = 2$

means: “variable x equals 2”

In Python,

$x = 2$

means: “assign the value 2 to variable x”

$x = 2$ is called an assignment statement

Declarations

Wait!

Where is the type declaration?

In other languages:

x : integer	*(Pascal)*
int x;	*(C#)*

In Python, no variable declarations are needed. The Python interpreter infers what is the type of each expression.

Python is a dynamically-typed language

Declarations

Variables do not need a declaration

```
a = 1          # interpreter infers a has type integer
b = 5          # interpreter infers b has type integer
c = a + b      # interpreter infers c has type integer
```

Variables and Types

Variables **must be created** before they can be used

```
print(t)
Traceback (innermost last):
File "<interactive input>", line 1, in
<module>NameError: name 't' is not
defined
```

Variables and Types

```
x = 2                # interpreter sets the value, type, and id

print(x)             # print value
 2

print(type(x) )      # don't print x's value but its type
<class 'int'>
```

Object types

Objects always have a type

```
a = 1
print (type(a))
<class 'int'>

a = "Hello"
print (type(a))
<class 'str'>

print (type(1.0) )
<class 'float'>
```

UNIVERSITY OF TWENTE.

Built-in types

Every language comes with some predefined things.

In Python, there are built-in object types for:

- numbers (type can be integer or float)
- text (type is string)
- truth values (type is boolean)
- and a number of other things ...

UNIVERSITY OF TWENTE.

Numeric types

• integer	2 -23 +10045654
• float	2.71 1e-3

Literals

A literal is an actual value typed out.

- A *variable* is an expression that references a value through the identifier
- A *literal* is an expression that represents the value itself

```
x = 2  # the variable is x and the literal is 2
       # afterwards, variable x has the same value as
       # literal 2
```

Variables obtain their type and value from the expression assigned to them

String literals

String literals are written in single quotes or double quotes:

'xyzzy'

"frobozz"

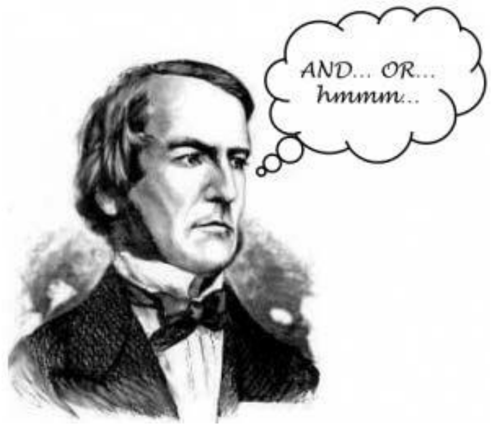
It doesn't matter which quote character you use, as long as opening and closing quote are the same.

Homework: how do you put a quote character inside a string literal?

Some other types

Boolean values:

```
x = True
print (type(x) )
<class 'bool'>
```



George Boole (1815 – 1864)

```
  # boolean expressions can have only two values: True , False
```

List values:

```
x = [0, 1, 4, 9, 16, 25]
print (type(x) )
<class 'list'>
```

Valid names

Variable names can be arbitrarily long.

Some examples:

x

my_name

airspeed_of_unladen_swallow

Llanfairpwllgwyngyllgogerychwyrndrobwyll

Pragmatic rule: choose <u>meaningful</u> names!

Invalid names

Not all names are valid!

```
76trombones = 'big party'
SyntaxError: invalid syntax

more@ = 1000
SyntaxError: invalid syntax

class = 'Spatial analysis'
SyntaxError: invalid syntax
```

Python keywords

Python has 33 reserved names, keywords:

False None True and as assert break

class continue def del elif else

except finally for from global if import in is

lambda nonlocal not or pass

raise return try while with yield

Do not use these as variable names!

Naming problems

```
bad name = 5
SyntaxError: invalid syntax
```

(names cannot contain spaces!)

```
Bob = 23
year = bob
NameError: name 'bob' is not defined
```

(names are case-sensitive)

The Python language is case-sensitive

Remember

UNIVERSITY OF TWENTE.

Naming problems

Be careful with too obvious names!

```
print ( type(2) )
<class 'int'>
```

```
type = 23
print ( type(2) )
TypeError: 'int' object is not callable
```

Whoops! Existing things (like type) can be destroyed!

Do not name your variable with already existing function names!

Thanks for your attention