



INST0002 PROGRAMMING 1 & INST0091 INTRODUCTION TO PROGRAMMING

LECTURE 02 – VARIABLES, OPERATORS & EXPRESSIONS

Dr Daniel Onah
Lecturer
Module Leader

Copyright Note:

Copyright Licence of this lecture resources is with the Module Lecturer named in the front page of the slides. If this lecture draws upon work by third parties (e.g. Case Study publishers) such third parties also hold copyright. It must not be copied, reproduced, transferred, distributed, leased, licensed or shared with any other individual(s) and/or organisations, including web-based organisations, online platforms without permission of the copyright holder(s) at any point in time.

Recap

- Computer programming & Programs
- History of Python
- Data type , print and input functions
- Command line programming
- First Python program
- Syntax errors and comments

Re: Cap☺



Learning outcomes

At the end of this lecture student should be able to:

- define and use variables and constants
- understand the properties and limitations of integers and floating-point numbers
- appreciate the importance of comments and good code layout
- write arithmetic expressions and assignment statements
- create programs that read and process inputs, and display the results
- learn how to use Python typecasting on literals such as strings, integers, floats etc.

Outline

- Variables and constants
- Integers and floating-point numbers
- Comments and good code layout
- Arithmetic expressions and assignment statements
- Read and process inputs, and display the results
- Python strings
- Summary

Variables and Naming Convention

Variables

When programs execute a computation, we will want to store values so we could use them later. In `Python` program , we use **variable** to store values.

Definition:

By definition, a variable is a storage location in a computer program. Each variable has a **name** and **holds a value**.

Syntax `variableName = value`

Variables

Names of previously
defined variables

A variable is defined
the first time it is
assigned a value

`totalCount = 0`

`totalCount = bottles * BOTTLE_VOLUME`

The expression that replaces
the previous value

`totalCount = totalCount + cans * CAN_VOLUME`

The same name can
occur on both sides

Names of previously
declared variables

Constants

Constants are **fixed values** that **cannot** be changed. Fixed values such as numbers, letters and strings are known as constants because their values does not change.

Numeric constants for example:

```
>>> print(1234)
```

```
1234
```

String constants uses single quotes (‘ ’) or double quotes (“ ”)

```
>>> print('Hello world')
```

```
Hello world
```

Reserved Words

Reserved words are words that are known to the interpreter. They are special words built along-side with the compiler or interpreters for performing specific programming tasks.

You **cannot & should not** use **reserved words** as variable names/identifiers

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

Variables

A **variable** is a **named place in the memory** where a programmer can store data and later retrieve the data using the **variable “name”**

Programmers get to **choose** the names of the **variables**

You can change the contents of a **variable** in a later statement

Example:

x = 77.7

y = 99

Stored in memory

x

77.7

y

99

Variables

A **variable** is a **named place in the memory** where a programmer can store data and later retrieve the data using the **variable “name”**

Programmers get to **choose** the names of the **variables**

You can change the contents of a **variable** in a later statement

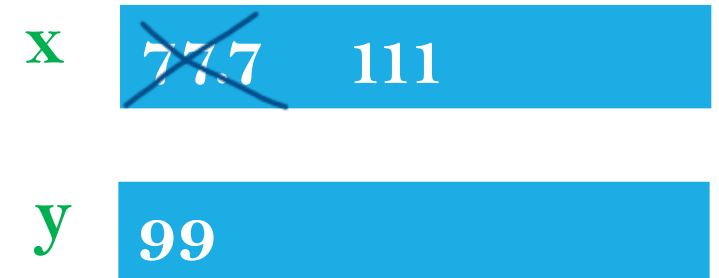
Example:

x = 77.7

y = 99

x = 111

Store in memory



Variable Name Rules

Variable names should start with a **letter** or **underscore** (_). It is case sensitive, and the key rule is that it **must consist** of **letters**, **numbers** and **underscores**.

Good:	Bad:	Different:
spam	21spam	spam
eggs	#sign	Spam
spam21	var.17	SPAM
_speed		

Variable Name Rules

Variable Name	Comment
<code>cubeVolume</code>	Variable names consist of letters, numbers , and underscore character.
<code>x</code>	In mathematics, you use short variable names such as z or y. This is also legal in Python, but not very common, because it can make programs harder to understand
<code>CubeVolume</code>	Caution: Variable names are case sensitive. This variable name is different from <code>cubeVolume</code> , and it violates the standard convention that variable names if two words should start with a lowercase letter (camel case).
<code>6pack</code>	Error: Variable names cannot start with number.
<code>cube Volume</code>	Error: Variable names cannot contain spaces.
<code>class</code>	Error: You cannot use a reserved word as a variable name.
<code>letter/us.ot</code>	Error: You cannot use symbols such as “/” or “.” in variable names

Sentences/Lines

`x = 2` ← Assignment statement

`x = x + 2` ← Assignment with expression

`print(x)` ← Print statement

Keys:

Variable

Operator

Constant

Function

Mnemonic Variable Names

There is a bit of **best practice** on how variable names are chosen.

We provide a name for our variable that would help us remember what we intend to store in them (“memory aid” = “**mnemonic**”)

Constants

A **constant variable** or simply a **constant** is a variable whose value **cannot be changed after it has been assigned and we have initialized the value**. It might generate a **syntax error** if you attempt to assign a new value to the variable. It is a common convention to specify a constant variable with the use of all **capital letters (UPPER CASE)** for its name. For example:

CUBE_VOLUME = 3.4

MAX_SIZE = 215

By following this convention, you provide information to your program that the variable in all **capital letters** to be a constant throughout the program.

Constants

It is a good programming style to use named constant in your program to explain numeric values. For example, compare the statements:

```
totalVolume = cube * 3.4
```

and

```
totalVolume = cube * CUBE_VOLUME
```

A programmer reading the first statement may not understand the significance or the meaning attributed to the number 3.4.

The second statement, with a **named constant**, makes the computation much clearer.

Expression and Assignment Statements

Expression

```
sql3z78ed = 49.0  
sql3z78pd = 17.9  
Sql3k9red = sql3z78ed * sql3z78pd  
print(Sql3k9red)
```

What is this bit of
code doing?

Expression

```
sql3z78ed = 49.0
```

```
sql3z78pd = 17.9
```

```
Sql3k9red = sql3z78ed * sql3z78pd
```

```
print(Sql3k9red)
```

```
a = 49.0
```

```
b = 17.9
```

```
c = a * b
```

```
print( c )
```

What is this bit of
code doing?

Expression

```
sql3z78ed = 49.0
```

```
sql3z78pd = 17.9
```

```
Sql3k9red = sql3z78ed * sql3z78pd
```

```
print(Sql3k9red)
```

```
a = 49.0
```

```
b = 17.9
```

```
c = a * b
```

```
print(c)
```

What is this bit of
code doing?

```
hours = 49.0
```

```
rate = 17.9
```

```
pay = hours * rate
```

```
print(pay)
```

Assignment Statements

We assign a value to a variable using the assignment statement (=).

An assignment statement consists of an **expression on the right-hand side** and a **variable** to store the result on the left-hand side.

x =

7.6 * x * (13 - x)

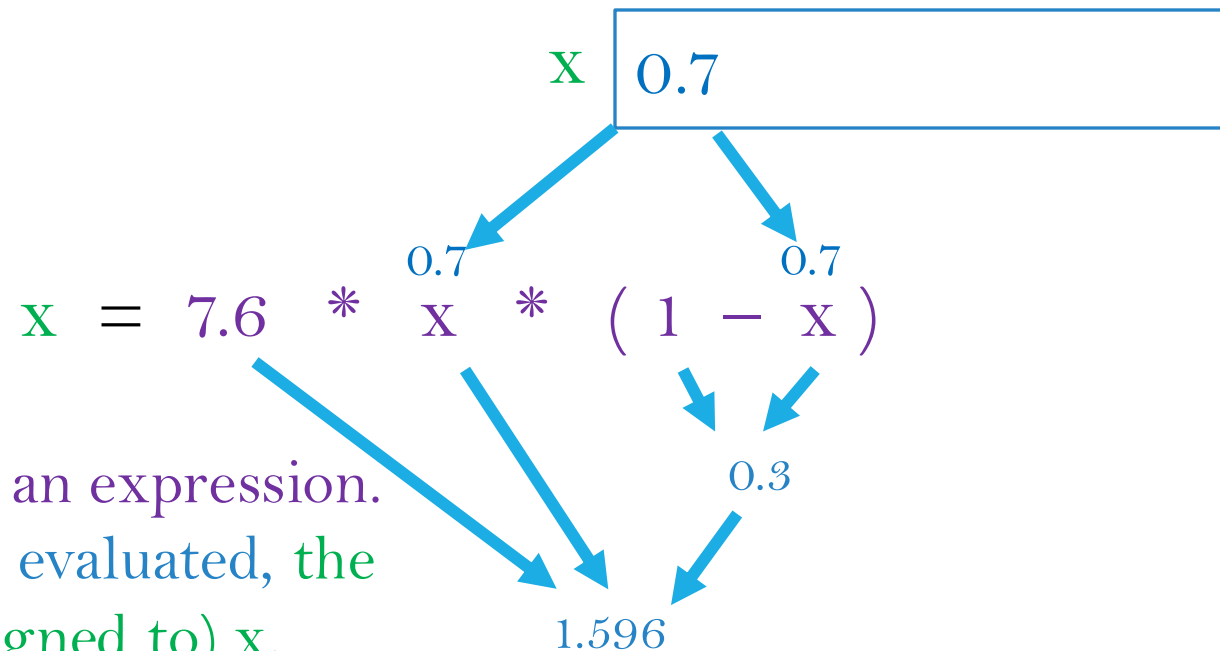
Remember a variable is a memory location used to store a value (e.g., floating value: 7.1).

The right side is an expression.

Once the expression is evaluated, the result is placed-in (assigned to) the variable on the left-hand side (i.e., x).

Example

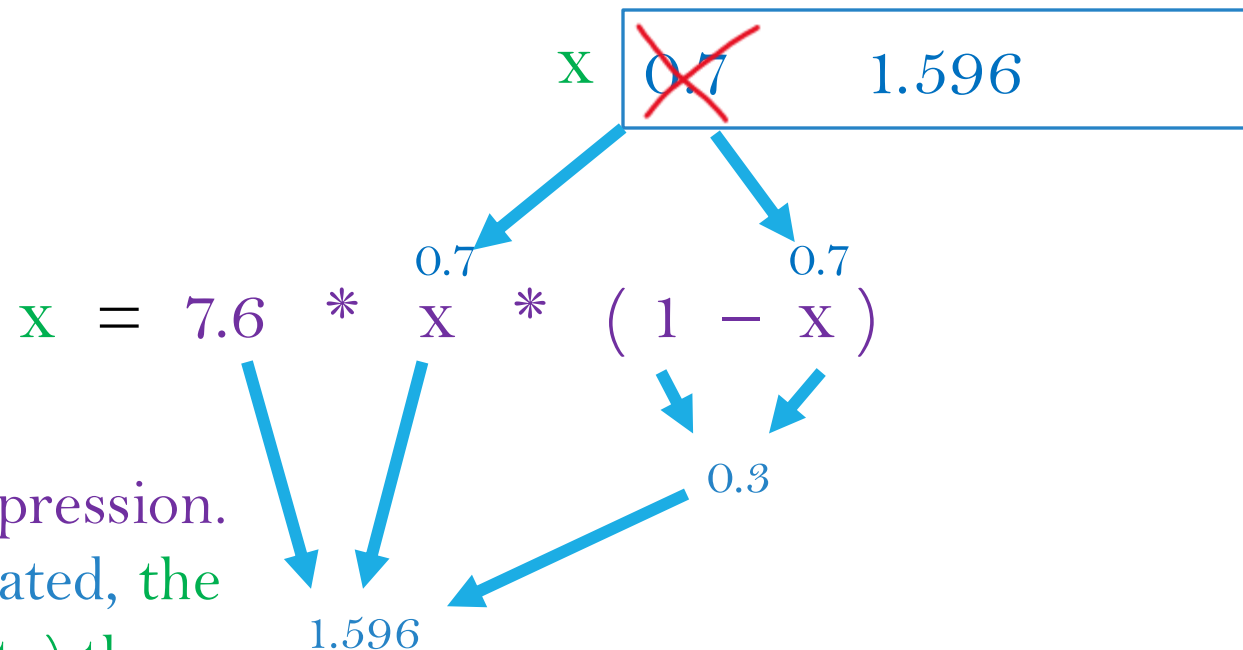
A variable is a memory location used to store a value (0.7). Let's say **x** stores a value 0.7.



The right-hand side is an expression. Once the expression is evaluated, the result is placed in (assigned to) **x**.

Example

A variable is a memory location used to store a value. The value stored in a variable can be updated by replacing the old value (0.7) with a new value (1.596).



The right-hand side is an expression. Once the expression is evaluated, the result is placed in (assigned to) the variable on the left-hand side (i.e., x).

Numeric Expression

Every programming language has some kind of way of doing numbers and math. Do not worry, we will not be covering deep mathematics in this module. You don't need to be a math genius to really do well in programming.

We will introduce you to some math symbols, that are needed to perform specific operations.

When formulae are used as an expression in Python, there is an order of **precedence** for the operation (e.g. BODMAS).

B: **Bracket**; O: **Of**; D: **Division**; M: **Multiplication**; A: **Addition**; S: **Subtraction**

Operators used in Numeric Math Expression

Operator	Operation
+	Addition/plus
-	Subtraction/minus
*	Multiplication/asterisk
/	Division/slash
**	Power/Exponential
%	Modulo/Remainder
<	Less-than
>	Greater-than
<=	Less-than-equal
>=	Greater-than-equal

Numeric Expression

Addition

```
>>> x = 21
```

```
>>> x = x + 4
```

```
>>> print(x)
```

```
25
```

Multiplication

```
>>> y = 202 * 15
```

```
>>> print(y)
```

```
3030
```

Numeric Expression

Division

```
>>> z = y / 100 # note the value of y is 3030
```

```
>>> print(z)
```

30.3

Modulo/Remainder

```
>>> j = 79
```

```
>>> k = j % 4
```

```
>>> print(k)
```

3

$$\begin{array}{r} 19 \text{ R } 3 \\ 4 \overline{) 79} \\ \underline{76} \\ 3 \end{array}$$

Check this using:

<https://www.wolframalpha.com/input>

Operators and Precedence

Operator Precedence

When we string operators together in an expression – Python must know which one to execute first.

This is called **operator precedence**. Which operator **takes precedence** over the others?

For example:

```
x = 1 + 2 * 3 - 4 / (5 ** 6)
```

Logic: BODMAS

Operator Precedence Rules

Highest precedence rule to lowest precedence rule:

Parentheses are always respected and considered first (bracket)

Exponentiation (raise to power of)

Division


Multiplication and Remainder

Addition

Subtraction

Left to right

Parenthesis
Power
Division
Multiplication
Addition
Subtraction
Left to Right



Other Operator Precedence

We can consider using a formula know as **BODMAS**.

B – Bracket

O – Of (Power of – exponential)

D – Division

M – Multiplication

A – Addition

S – Subtraction


Example

>>> `x = 3 + 6 ** 2 / 5 * 4`  **Green perform first**

>>> `print(x)`

Result: **31.8**

Parenthesis
Power
Division
Multiplication
Addition
Subtraction
Left to Right



Logic of BODMAS

$3 + 6 ** 2 / 5 * 4$

$3 + 36 / 5 * 4$

$3 + 7.2 * 4$

$3 + 28.8$

31.8

Operator Precedence

Parenthesis
Power
Division
Multiplication
Addition
Subtraction
Left to Right



Remember the rules from top to bottom

- When writing code – use parentheses
- When writing code – keep mathematical expressions simple enough that they are easy to understand
- Break all long series of mathematical operations/expressions up to smaller unit to make them more clear

Data Type in Python

Data Type

In Python **data values** that represent information can be of **different types** i.e. each value is of specific type.

The data type of a value **determines how the data is represented in the program** and what operations can be performed on that data.

A data type provided by specific programming language is called a **primitive data type**. Python supports quite a few data types: **numbers, text strings, files , containers**, and so on.

Data Type

In Python, there are several **different types of numbers**. An **integer** value e.g. **18** is a **whole number** without fractional part. When a fraction part is required such as **0.33**, we use **floating-point numbers**, which are called **float** in Python.

Data Type

Type	Description
Integers	Whole numbers: - 14, -2, 0, 1, 100, 123
Float	Floating point numbers with decimal: -2.5, 0.0, 74.8, 11.2

Data Type

Number Literals in Python

Number	Type	Remark
9	int	An integer has no fractional part
-9	int	Integers can be negative.
0	int	Zero is an integer
0.6	float	A number with fraction part has a type float.
7.0	float	An integer with a fractional part .0 has a type float
1E6	float	A number in exponential notation: 1×10^6 or 1000000.
2.85E-2	float	Negative exponential: $2.85 \times 10^{-2} = 2.85/100 = 0.0285$
200, 000		Error: Do not use a comma as a decimal separator
3 ½		Error: Do not use fractions; use decimal notation: 3.5

Data Type

In Python all variables, literals (*Strings, int, float* etc.) and constants have a **type**

Python knows the **difference** between an integer number and string

Let's look at an example, “+” sign could be use for two purposes as **operator overloaded**. It could be used as “**addition**” if we have numbers and “**concatenate**” if we have strings.

```
addNum = 3 + 2
print(addNum)
output:
5
```

Data Type

A variable in Python can store any type. The data type is explicitly *associated with the assigned value*, and not the variable itself. For example, let's consider a variable '**cost**' below of type **float**:

```
cost = 10.15
```

The same variable can later hold a value of type **int**:

```
cost = 17
```

Concatenation

Concatenation is the process of joining two or more strings together.

```
greetings = 'Hello' + 'Danny'  
print(greetings)  
output:  
HelloDanny
```

That is, we could say: concatenate = glue or put together

Data Type Matters!

Python know what “**type**” every variables, literals and constant are:

- Some operations are not allowed (**prohibited**)
- For example: You cannot “add number 1” to a string.

```
greetings = 'Hello' + 'Danny'  
greetings = greetings + 1
```

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: can only concatenate str (not "int") to str

Identifying Data Type

We can identify data type by asking Python what type is a variable, literal or constant by passing this into a type function: `type()`

```
greetings = 'Hello' + 'Danny'  
print(type(greetings))
```

Output:

```
<class 'str'>
```

Identifying Data Type

We can identify data type by asking Python what type is a variable, literal or constant by passing this into a type function: `type()`

```
print(type('Hello'))
```

Output:

```
<class 'str'>
```

Identifying Data Type

We can identify data type by asking Python what type is a variable, literal or constant by passing this into a type function: `type()`

```
print(type(13))
```

Output:

```
<class 'int'>
```

Identifying Data Type

We can identify data type by asking Python what type is a variable, literal or constant by passing this into a type function: `type()`

```
print(type(97.3))
```

Output:

```
<class 'float'>
```


Type Conversion Or TypeCasting

When you have an expression with both integer and typecasting a floating point, the integer is **implicitly** converted to a floating-point number.

```
print(float(111) + 11)
```

Output:

122.0

Casting is the process of changing a **variable data type** from one **type** to **another**.

You can perform typecasting with Python built-in functions such as **int()**, **float()**, **str()**, **complex()** etc.

Type Conversion Or TypeCasting

Casting is the process of changing a **variable data type** from one **type** to **another**.

You can perform casting with Python built-in functions such as `int()`, `float()`, `str()`, `complex()` etc.

```
i = 59
f = float(i)
print(f)
```

Output:

59.0

Strings Conversion

Strings

In Python a string is made up of combination of a sequence of characters or letters. A string can be executed from the standard python built-in `input()` function, or we can define this in a single or double quotes. We can also repeat a string by multiplying by the number of times using an integer. Two or more strings can be concatenated together.

- We can find the length of a string of characters e.g. by using the python in-built `len()` function
- All objects in Python can also be converted to a string e.g. we can convert numbers to string `str(5 ** 2)`
- We can use string with slices. A slice produces from a given string one character or some fragment in case of substring.

For example:

```
greetings = 'Hello'
print(greetings[0])
```

Output: **H**

Slices

A slice gives from a string one character in the position of the index. E.g. `S[i]` means the 'i' is the position index of the string character. In Python there is no data type for characters of the string. The type of `S[i]` also hold the type of the string (**str**) as the source string.

If we specify a negative index, then it counted from the end of the string starting with the number **-1**.

Note that if the index of the slice `greeting[i]` is greater than or equal to the `len(greetings)`, or less than `-len(greeting)`, this will lead to the following errors:

IndexError: string index out of rang

```
greetings = 'Hello'
print(greetings[-1])
```

Output: **o**

Strings Conversion

You can also use `int()` and `float()` to convert between *strings* and *integers*.

You will get an **error** if the string does not contain numeric characters.

```
val = '1233'  
print(type(val))
```

Output:

```
<class 'str'>
```

Strings Conversion

You can also use `int()` and `float()` to convert between *strings* and *integers*.

You will get an **error** if the string does not contain numeric characters.

```
val = '1233'  
print(val + 1)
```

Output:

Traceback (most recent call last):

File "<stdin>", line 2, in <module>

TypeError: can only concatenate str (not "int") to str

Strings Conversion

You can also use `int()` and `float()` to convert between *strings* and *integers*.

You will get an **error** if the string does not contain numeric characters.

```
val = '1233'  
ival = int(val)  
print(type(ival))  
print(ival + 1)
```

Output:

```
<class 'int'>
```

```
1234
```


Strings Conversion

You can also use `int()` and `float()` to convert between *strings* and *integers*.

You will get an **error** if the string does not contain numeric characters.

```
greetings = 'Hello Danny'  
greet = int(greetings)
```

Output:

Traceback (most recent call last):

File "<stdin>", line 2, in <module>

greet = int(greetings)

ValueError: invalid literal for int() with base 10: 'Hello Danny'

User Input

We can instruct Python to take a user input, pause and read data input from the user by using the input function: `input()`

The `input()` function returns a string.

```
name = input('what is your name?: ')  
print('Welcome', name, '!')
```

Output:

```
what is your name?: Danny  
Welcome Danny !
```

Strings Example

Let's look at a program that prints a pair of initials.

```
name1 = 'Danny'  
name2 = 'Onah'  
initials = name1[0] + "." + name2[0] + "."  
print(" Your initials are: ", initials)
```

Output:

Your initials are: D . O.

Developing a Program

Developing a Program

The program we write in `Python` follow through a `standard syntax convention`. Therefore, care should be taken while writing you programs.

It is important to note that, for a good program construct, there is a need for `comments to describe what each line of codes or constructs are doing`.

Comments in Python

Any statement or construct after the `#` key is **ignored by Python**

Why do we comment our codes?

- To describe what is going to happen in a sequence of code
- To document who is the author of the code or other ancillary/support information to help with the understanding of the program
- To disable or turn off a line of code – perhaps temporarily

Converting User Input

In Python, if we want to read an **integer** number from a user, we must convert the input from a string to integer using an `int` type conversion function.

```
# Convert elevator floor  
floor = input('Enter floor:')  
userFloor = int(floor) + 1  
print('User is in floor:', userFloor)
```

Output:

Enter floor: 2

User is in floor: 3





Variable Naming Convention

Traditional naming style with mixed of capital and small letters.

Home Exercise

Write a program to prompt the user for hours and rate per hour to compute gross pay.

Expected outcome:

Enter Hours: 36.5

Enter Rate: 8.2

Pay: £299.3

Summary

Summary

- We discussed variables
- Reserved words specific to Python interpreter
- We looked at assignment statements
- We discussed the various data types
- We looked at operators and operator precedence
- We looked at conversion between types
- We discussed user input and comments in programs



What to do Next

- Make sure you're familiar with how to enter, save and run a program.
- Check out the introductions for both the teaching concepts and coding strands.
- Check that you're familiar with where things are on the Moodle site and how to navigate around the resources.
- Practice-practice-practice! your Python programs using editors and command line.
- Start working on your tutorial sheet for the week.

Further Reading

You can read further this week's lecture from the following chapters:

- Python for Everyone (3/e) : By **Cay Horstmann & Rance Necaise**
 - Chapter 2 Programming with Numbers and Strings
- Learning Python (5th Edition): By **Mark Lutz**
 - Part 2 Types and Operations – Chapter 5 Numeric Types
- Basic Core Python Programming: By **Meenu Kohli**
 - Chapter 2 Python Basics & Chapter 3 Numbers, Operators and In-built functions

Next Lecture 3

Lecture 3 - Control flow conditional statements, Loops and Iteration