PYTHON

YT0744

2ND PHASE PROFESSIONAL BACHELOR ELECTRONICS-ICT

Johan Van Bauwel

E-mail: Johan.vanbauwel@thomasmore.be

Office: A114

COURSE

- Theory:
  - 12 x 2hr sessions
  - First sessions by me, other sessions presented by you (about ½ hr – 1 hr max per session, rest: exercises)
- Labs:
  - 12 x 2hr sessions
  - Lab project
- Study time:
  - Theory: 3SP x 25hrs/SP = 75hrs
  - Labs: 3SP x 25hrs/SP = 75hrs

  Total: 150hrs (about 19 full-time days)

COURSE

- Course objectives: at the end of the course, students will be able to:
  - Explain the topics covered during this course
  - Design & program algorithms for solving problems using Python
  - Make use of libraries, functions, classes, loops, flow control, lists, tuples, dictionaries, string manipulation, file I/O, regular expressions, organizing files, work with CSV and JSON data, …

(non-exhaustive list)

COURSE

- Exam:
  - Theory: 25% topic presentation, 75% written exam during exam period
  - Lab: 50% project, 50% practical lab exam (closed-book) during last lab session(s)

  August/September:
  - Theory: 100% written exam during exam period
  - Lab: 50% project, 50% practical lab exam (closed-book) during exam period

PYTHON BASICS

“The **Python** programming language has a wide range of **syntactical** constructions, **standard library functions**, and **interactive development environment** features.
Fortunately, you can ignore most of that;

you just need to learn enough to write some handy little programs”.

This chapter has a few **examples** that encourage you to type into the **interactive shell**, which lets you execute Python instructions one at a time and shows you the results instantly.

Using the interactive shell is great for learning what basic Python instructions do, so give it a try as you follow along.

You’ll remember the things you do much better than the things you only read.

ENTERING BASIC EXPRESSIONS

Launch IDLE, which you installed with Python in the introduction.

On Windows, open the Start menu, select **All Programs** ▸ **Python 3.3**, and then select **IDLE (Python GUI)**.
On OS X, select **Applications** ▸ **MacPython 3.3** ▸ **IDLE**.
On Ubuntu, open a new Terminal window and enter **idle3**.

```
>>> 2 + 2
4
```

```
Python 3.3.2 (v3.3.2:d047928ae3f6, May 16 2013, 00:06:53) [MSC
v.1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 2 + 2
4
```

```
>>> 2
2
```

Python ‘evaluates’ “Expressions” containing “Operators”

ERRORS ARE OKAY!

Programs will **crash** if they contain code the computer can’t understand, which will cause Python to show an error message.

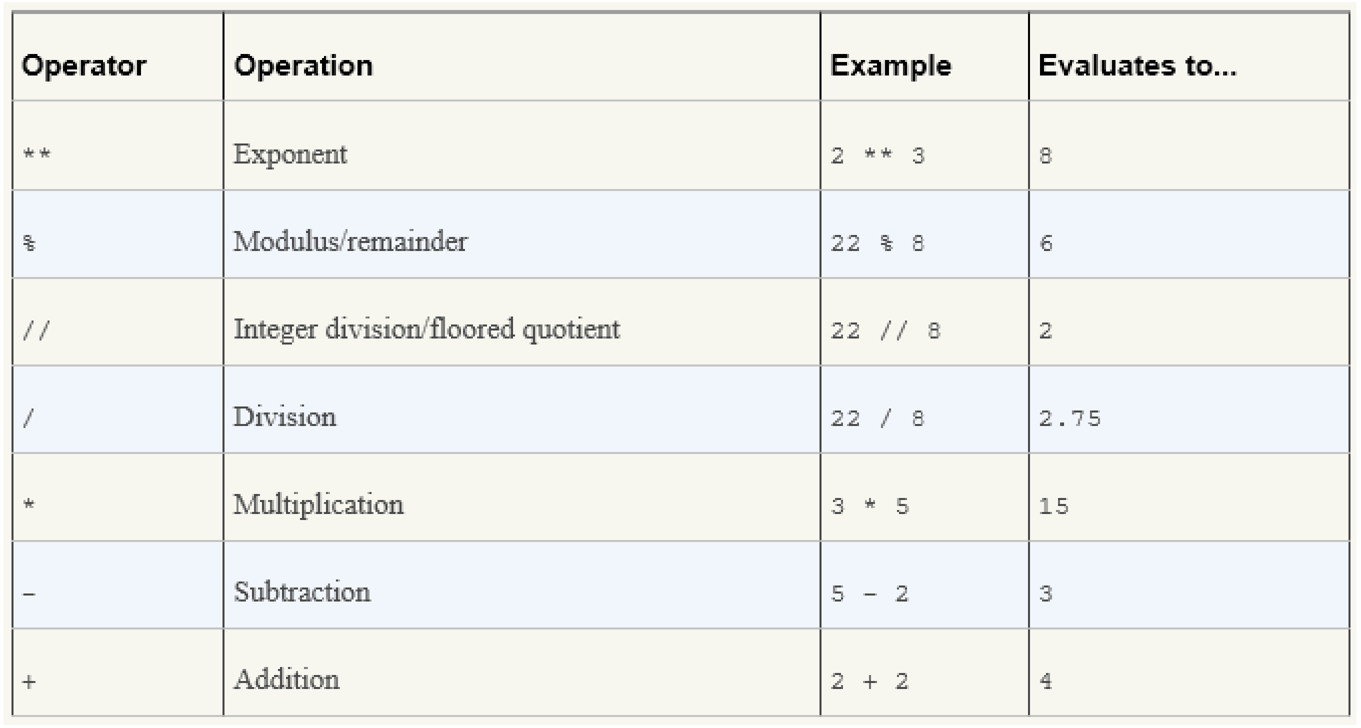
An error message won’t break your computer, though, so don’t be afraid to make mistakes.

A ***crash*** just means the program stopped running **unexpectedly**.

If you want to know more about an error message, you can search for the exact message text online to find out more about that specific error.

You can also check out the resources at ***http://nostarch.com/automatestuff/*** to see a list of common Python error messages and their meanings.

MATH OPERATORS



Operator	Operation	Example	Evaluates to...
`**`	Exponent	`2 ** 3`	`8`
`%`	Modulus/remainder	`22 % 8`	`6`
`//`	Integer division/floored quotient	`22 // 8`	`2`
`/`	Division	`22 / 8`	`2.75`
`*`	Multiplication	`3 * 5`	`15`
`-`	Subtraction	`5 - 2`	`3`
`+`	Addition	`2 + 2`	`4`

The ** operator is evaluated first; the *, /, //, and % operators are evaluated next, from left to right; and the + and - operators are evaluated last (also from left to right).

MATH OPERATORS SIMPLE EXAMPLE

```
>>> 2 + 3 * 6
20
>>> (2 + 3) * 6
30
>>> 48565878 * 578453
28093077826734
>>> 2 ** 8
256
>>> 23 / 7
3.2857142857142856
>>> 23 // 7
3
>>> 23 % 7
2
>>> 2 + 2
4
>>> (5 - 1) * ((7 + 1) / (3 - 1))
16.0
```

$$\left\{\begin{array}{l}(5 - 1) * ((7 + 1) / (3 - 1)) \\ 4 * ((7 + 1) / (3 - 1)) \\ 4 * (8) / (3 - 1)) \\ 4 * (8) / (2) \\ 4 * 4.0 \\ 16.0\end{array}\right.$$

PARSE ERRORS

These rules to form expressions are a fundamental in the Python as a language.

This is a grammatically correct English sentence.
This grammatically is sentence not English correct a.

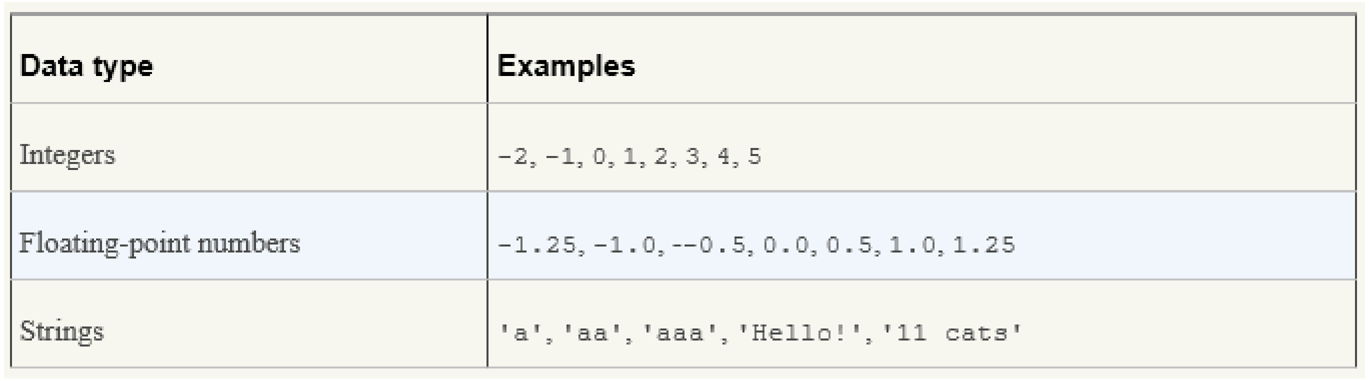
Second line is difficult to **parse** since it doesn't follow the rules of English.

Python won't be able to understand 'bad instructions' and will display a SyntaxError error message, as shown here:

```
>>> 5 +
File "<stdin>", line 1
5 +
^
SyntaxError: invalid syntax
>>> 42 + 5 + * 2
File "<stdin>", line 1
42 + 5 + * 2
^
SyntaxError: invalid syntax
```

INTEGER, FLOAT AND STRING TYPES

A *data type* is a category for values, and every value belongs to exactly one data type



Data type	Examples
Integers	`-2, -1, 0, 1, 2, 3, 4, 5`
Floating-point numbers	`-1.25, -1.0, --0.5, 0.0, 0.5, 1.0, 1.25`
Strings	`'a', 'aa', 'aaa', 'Hello!', '11 cats'`

The values -2 and 30, for example, are said to be *integer* values.
The integer (or *int*) data type indicates values that are whole numbers.
Numbers with a decimal point, such as 3.14, are called *floating-point numbers* (or *floats*).

Note that even though the value 42 is an integer, the value 42.0 would be a floating-point number.

STRING CONCATENATION

The meaning of an operator may change based on the data types of the values next to it.

The + is the add operator when it operates on two ints or floats.

However, when + is used on two string values, it joins the strings as the ***string concatenation*** operator.

```
>>> 'Alice' + 'Bob'
'AliceBob`
```

```
>>> 'Alice' + 42
Traceback (most recent call last):
File "<pyshell#26>", line 1, in <module>
'Alice' + 42
TypeError: Can't convert 'int' object to str implicitly
```

Your code will have to explicitly convert the integer to a string with str() function, because Python cannot do this automatically!

STRING REPLICATION

The * operator is used for multiplication int/float values.

But when the * operator is used on **one string value and one integer** value, it becomes the ***string replication*** operator.

```
>>> 'Alice' * 5
'AliceAliceAliceAliceAlice�much
```

```
>>> 'Alice' * 'Bob'
Traceback (most recent call last):
File "<pyshell#32>", line 1, in <module>
'Alice' * 'Bob'
TypeError: can't multiply sequence by non-int of type 'str'
>>> 'Alice' * 5.0
Traceback (most recent call last):
File "<pyshell#33>", line 1, in <module>
'Alice' * 5.0
TypeError: can't multiply sequence by non-int of type 'float'
```

Python wouldn’t understand these expressions: Multiply two words, and replicate an arbitrary string a fractional number of times ???

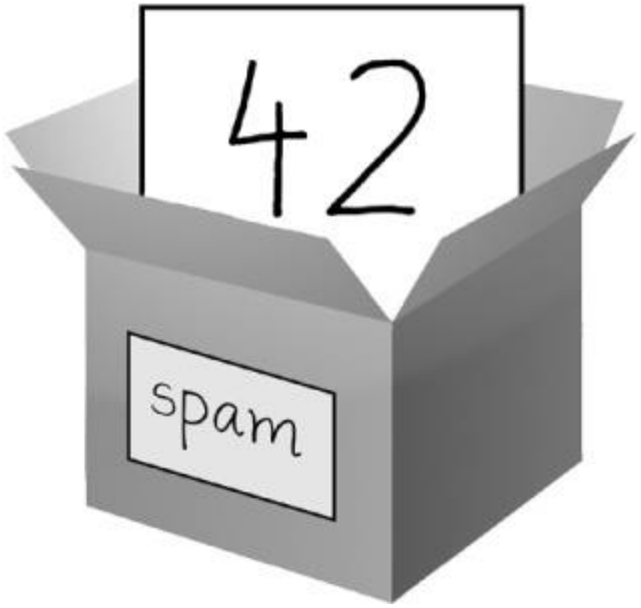
STORING VARIABLES

A ***variable*** is like a box in the computer’s memory where you can store a **single** value.

You’ll store values in variables with an ***assignment statement***.

An assignment statement consists of a **variable name**, an **equal sign** (called the *assignment operator*), and the **value** to be stored.

```
>>> spam = 40
>>> spam
40
>>> eggs = 2
v >>> spam + eggs
42
>>> spam + eggs + spam
82
>>> spam = spam + 2
>>> spam
42
```



VARIABLE INITIALIZATION/OVERWRITING

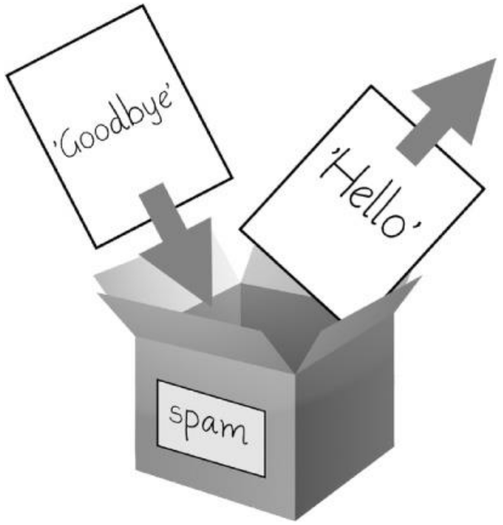
A variable is ***initialized* (or created)** the first time a value is stored in it.

Then, you can use it in expressions with other variables and values.

When a variable is assigned a new value w, the old value is forgotten, which is why spam evaluated to 42 instead of 40 at the end of the example.

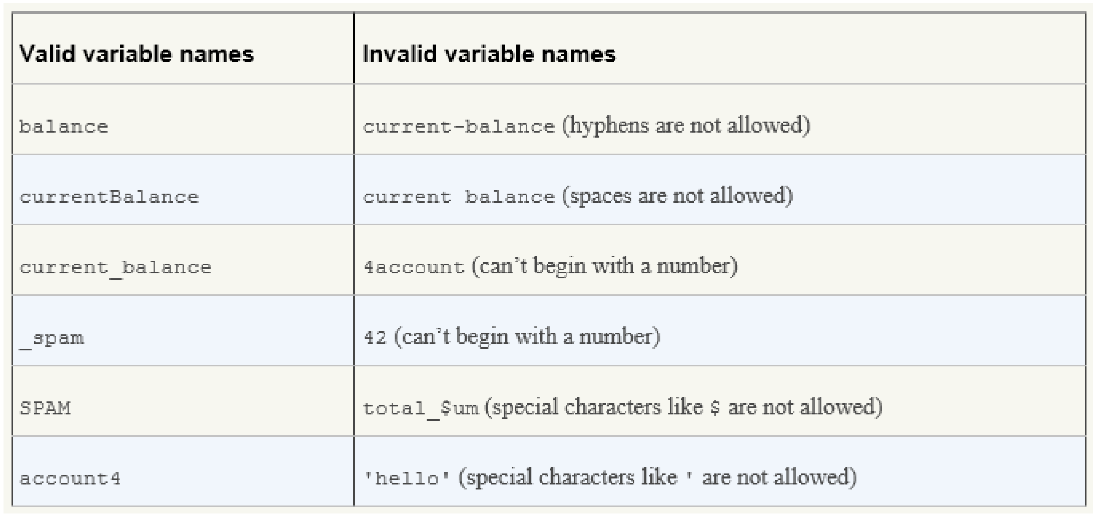
This is called ***overwriting*** the variable

```
>>> spam = 'Hello'
>>> spam
'Hello'
>>> spam = 'Goodbye'
>>> spam
'Goodbye'
```



VARIABLE NAMES

1. It can be only one word.
2. It can use only letters, numbers, and the underscore (_) character.
3. It can’t begin with a number.



Valid variable names	Invalid variable names
`balance`	`current-balance` (hyphens are not allowed)
`currentBalance`	`current balance` (spaces are not allowed)
`current_balance`	`4account` (can’t begin with a number)
`_spam`	`42` (can’t begin with a number)
`SPAM`	`total_$um` (special characters like `$` are not allowed)
`account4`	`'hello'` (special characters like `'` are not allowed)

spam, SPAM, Spam, and sPaM are four different variables.

It is a Python **convention** to start your variables with a lowercase letter.

This course uses **camelcase** for variable names instead of underscores; that is, variables lookLikeThis instead of looking_like_this.

YOUR FIRST PYTHON PROGRAM

```
# This program says hello and asks for my name.
print('Hello world!')
print('What is your name?') # ask for their name
myName = input()
print('It is good to meet you, ' + myName)
print('The length of your name is:')
print(len(myName))
print('What is your age?') # ask for their age
myAge = input()
print('You will be ' + str(int(myAge) + 1) + ' in a year.')
```

Entered your source code and save it as **hello.py**.

To execute your program. Select **Run ▸ Run Module** or just press the **F5** key.

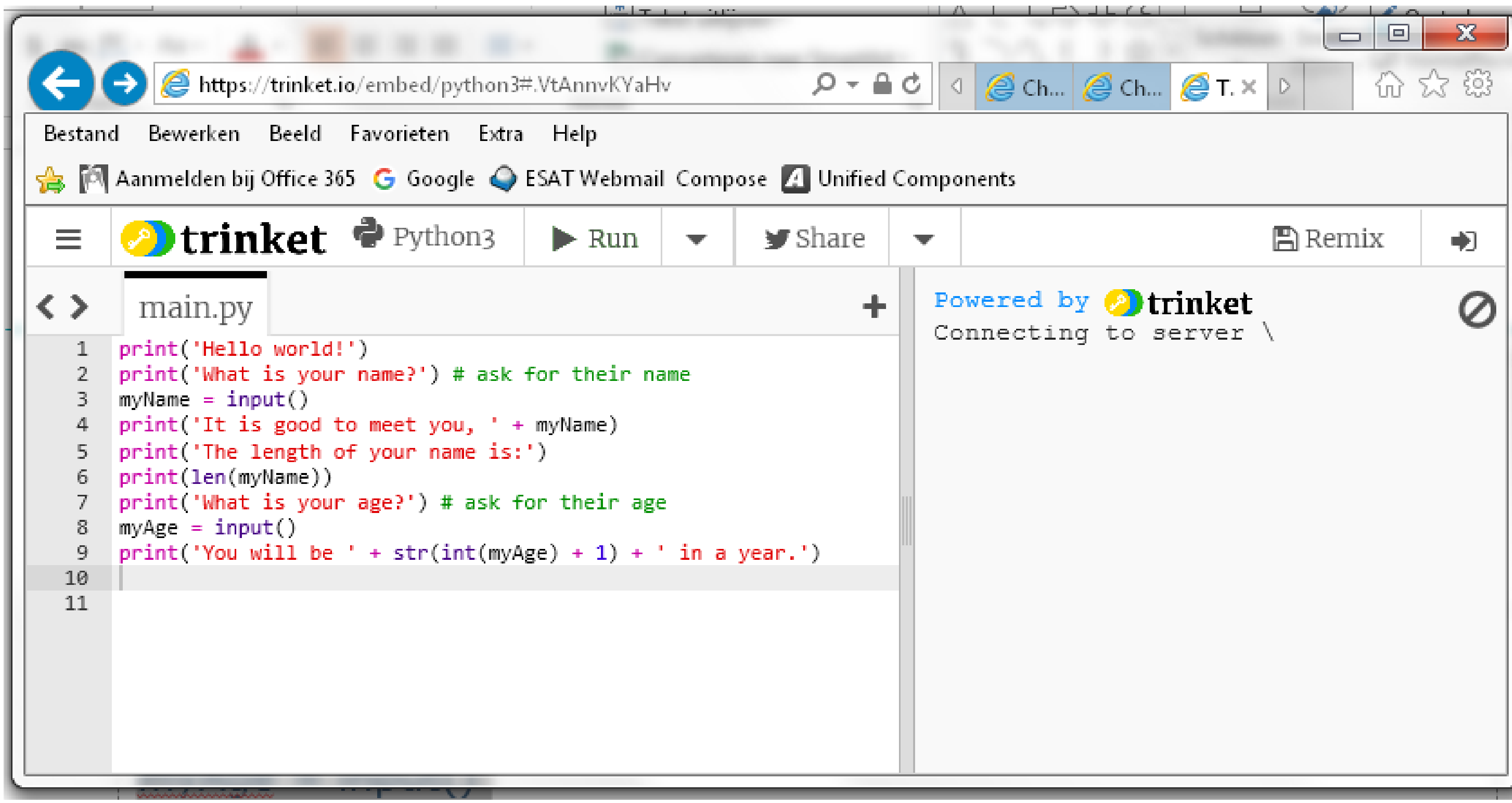
Your program should run in the **interactive shell window** that appeared when you first started **IDLE**. Remember, you have to press F5 from the file editor window, not the interactive shell window.

Enter your name when your program asks for it.

YOUR FIRST PYTHON PROGRAM IDLE RESULT

```
Python 3.3.2 (v3.3.2:d047928ae3f6, May 16 2013, 00:06:53)
[MSC v.1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ============ RESTART
========================
>>>
Hello world!
What is your name?
Al
It is good to meet you, Al
The length of your name is:
2
What is your age?
4
You will be 5 in a year.
>>>
```

YOUR FIRST PYTHON PROGRAM ON TRINKET


https://trinket.io/embed/python3#.VtAnnvKYaHv
Bestand Bewerken Beeld Favorieten Extra Help
Aanmelden bij Office 365 Google ESAT Webmail Compose Unified Components
trinket Python3 Run Share Remix
main.py

```python
print('Hello world!')
print('What is your name?') # ask for their name
myName = input()
print('It is good to meet you, ' + myName)
print('The length of your name is:')
print(len(myName))
print('What is your age?') # ask for their age
myAge = input()
print('You will be ' + str(int(myAge) + 1) + ' in a year.')
```

```
Powered by trinket
Connecting to server \
```

DISSECTING PROGRAM: COMMENT

```
# This program says hello and asks for my name.
```

Python ignores **comments**, and you can use them to write notes or remind yourself what the code is trying to do. Any text for the rest of the line following a hash mark (#) is part of a comment.

Sometimes, programmers will put a # in front of a line of code to temporarily remove it while testing a program. This is called ***commenting out*** code, and it can be useful when you're trying to figure out why a program doesn't work. You can remove the # later when you are ready to put the line back in.

Python also ignores the blank line after the comment. You can add as many blank lines to your program as you want.

DISSECTING PROGRAM: PRINT AND INPUT

```
print('Hello world!')
print('What is your name?')                    # ask for their name
print('It is good to meet you, ' + myName)
```

The line print('Hello world!') “Print out the text in the string 'Hello world!'.”

When Python executes this line, you say that Python is ***calling*** the print() function and the string value is being *passed* to the function. A **value** that **is** passed to a function call is an ***argument***.

Quotes are not printed to the screen. They just mark where the string begins and ends; they are not part of the string value.

```
myName = input()
```

This **function call evaluates** to a **string** equal to the user’s text, and the previous line of code assigns the myName variable to this string value.

DISSECTING PROGRAM: LEN() FUNCTION

```
print(len(myName))
```

You can pass the len() function a **string value** (or a variable containing a string), and the function **evaluates to the integer value** of the **number of characters** in that string.

```
>>> len('hello')
5
>>> len('My very energetic monster just scarfed nachos.')
46
>>> len('')
0
>>> print('I am ' + 29 + ' years old.')
Traceback (most recent call last):
File "<pyshell#6>", line 1, in <module>
print('I am ' + 29 + ' years old.')
TypeError: Can't convert 'int' object to str implicitly
```

Python gives an **error** because you can use the + operator only to add two integers together or concatenate two strings!

STR() FUNCTION

If you want to concatenate an integer such as 29 with a string to pass to print(), you'll need to get the value '29', which is the string form of 29.

```
>>> str(29)
'29'
>>> print('I am ' + str(29) + ' years old.')
I am 29 years old.
```

Because str(29) evaluates to '29', the expression 'I am ' + str(29) + ' years old.' evaluates to 'I am ' + '29' + ' years old.', which in turn Evaluates to 'I am 29 years old.'.

This is the value that is passed to the print() function.

```
print('You will be ' + str(int(myAge) + 1) + ' in a year.')
```

STR(), INT() AND FLOAT() FUNCTIONS

The **str()**, **int()**, and **float()** functions will evaluate to the **string**, **integer** and **floating-point** forms of the value you pass, respectively.

```
>>> str(0)
'0'
>>> str(-3.14)
'-3.14'
>>> int('42')
42
>>> int('-99')
-99
>>> int(1.25)
1
>>> int(1.99)
1
>>> float('3.14')
3.14
>>> float(10)
10.0
```

INPUT AND INT() FUNCTION

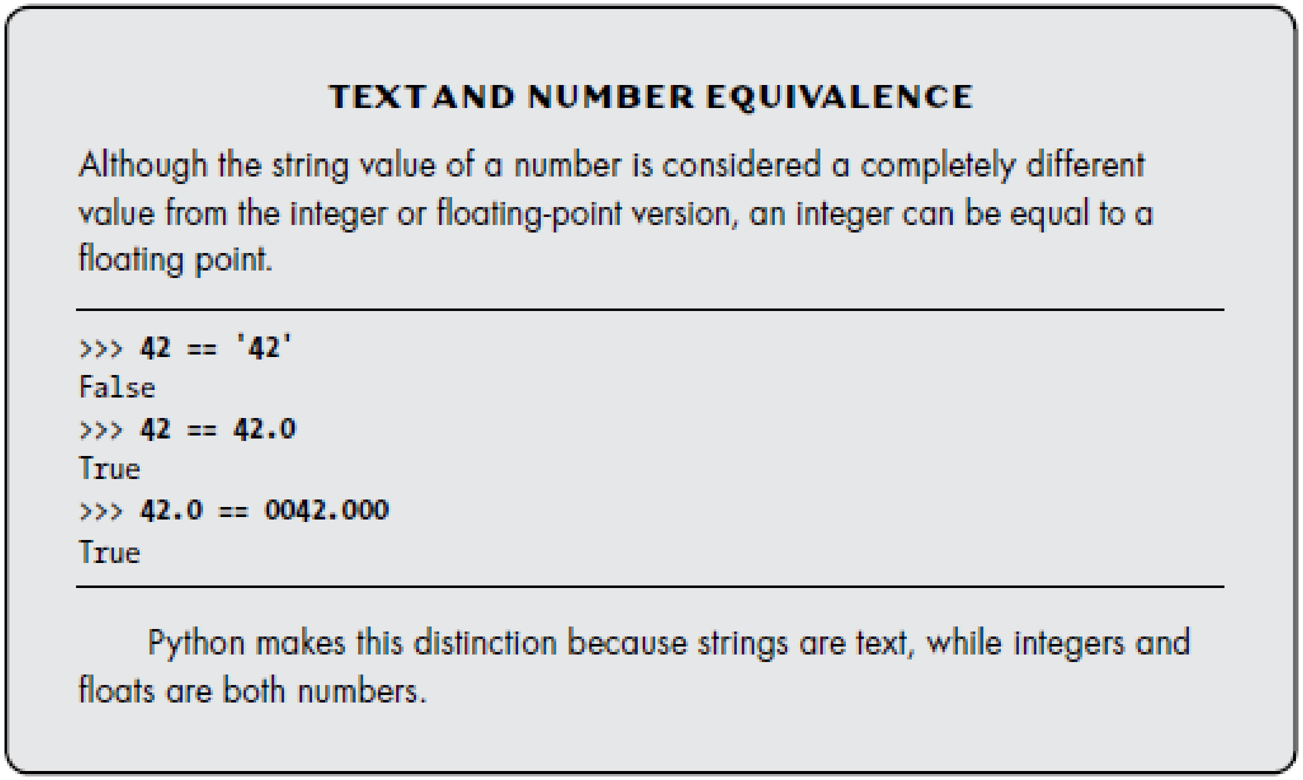
The **int()** function is also helpful if you have a **number as a string value** that you want to **use** in some **mathematics**.

```
>>> spam = input()
101
>>> spam
'101‘
>>> spam = int(spam)
>>> spam
101
>>> spam * 10 / 5
202.0
```

```
>>> int('99.99')
Traceback (most recent call last):
File "<pyshell#18>", line 1, in <module>
int('99.99')
ValueError: invalid literal for int() with base 10: '99.99'
>>> int('twelve')
Traceback (most recent call last):
File "<pyshell#19>", line 1, in <module>
int('twelve')
ValueError: invalid literal for int() with base 10: 'twelve‘

>>> int(7.7)
7
>>> int(7.7) + 1
8
```

TEXT AND NUMBER EQUIVALENCE



TEXT AND NUMBER EQUIVALENCE

Although the string value of a number is considered a completely different value from the integer or floating-point version, an integer can be equal to a floating point.

```
>>> 42 == '42'
False
>>> 42 == 42.0
True
>>> 42.0 == 0042.000
True
```

Python makes this distinction because strings are text, while integers and floats are both numbers.

SUMMARY

Expressions, and their component values—**operators**, **variables**, and **function calls**—are the basic building blocks that make programs.

Different types of operators (**+**, **-**, *****, **/**, **//**, **%**, and ****** for math operations, and **+** and ***** for string operations) and the **three data types** (**integers**, **floating-point numbers**, and **strings**)

The **print()** and **input()** functions handle simple text output and input (from the keyboard). The **len()** function takes a string and evaluates to an int of the number of characters in the string. The **str()**, **int()**, and **float()** functions will evaluate to the string, integer, or floating-point number form of the value they are passed.

In the **next chapter**, you will learn how to tell Python to make intelligent decisions about what code to run, what code to skip, and what code to repeat based on the values it has. This is known as ***flow control***, and it allows you to write programs that make intelligent decisions.

PRACTICE QUESTIONS

Which of the following are operators, and which are values?

```
*
'hello'
-88.8
-
/
+
5
```

What does the variable bacon contain after the following code runs?

```
bacon = 20
bacon + 1
```

What should the following two expressions evaluate to?

```
'spam' + 'spamspam'
'spam' * 3
```

Why does this expression cause an error? How can you fix it?

```
'I have eaten ' + 99 + ' burritos.'
```

THANKS & QUESTIONS

Thank you
for your
attention!

Questions?