CS115

INTRODUCTION TO PROGRAMMING IN PYTHON

CS 115 – Finding Information

1. Course Management System (Moodle)
   - Weekly Course Outline
   - Course Information
   - Lecture Notes and Code Examples
2. Course Information
   - Assessments and their weights
3. Weekly Course Outline
   - Lab dates
   - Weekly topic outline

Lab sessions

There are 10 lab sessions this semester.

During each lab session you will complete a lab assignment. Lab assignments must be completed individually.

Teaching assistants and undergraduate tutors will be present for the entire lab session.

You must attend the lab sessions and meet with the TA at the end of the lab session to receive a grade.

The teaching assistants will check and grade your assignment. They may ask questions about your code, ask you to make changes, etc.

There are no make-ups for lab sessions even with a medical report or special permission, however your lab average will be calculated by discarding your lowest lab grade.

Python Language and Modules

In class the exercises and examples will use the most recent version of Python from the Anaconda distribution (currently 3.9).

We will cover additional packages not included with standard Python, which are:

- `numpy`: package for scientific computing in Python.
- `matplotlib`: comprehensive library for creating static, animated, and interactive visualizations in Python.

Recommended Software

For the course, we recommend the Anaconda Individual Python 3.9 distribution.

Anaconda contains:

- **Python:** a *programming language* in which we write computer programs.
- **Python packages:** For scientific computing and computational modelling, we need additional libraries (*packages*) that are not part of the Python standard library. These allow us to create plots, operate on matrices, and use specialized numerical methods. Eg. numpy, pyplot, etc.
- **Spyder:** The name SPyDER derives from "Scientific Python Development EnviRonment" (SPYDER). It is an integrated development environment (IDE) for the Python language with advanced editing, interactive testing, debugging features.
- **Jupyter Notebook:** an alternative tool to develop open-source software, open-standards, and services for interactive computing across many programming languages.

Note about Software for CS115

We recommend the Anaconda distribution for ease of use.

It comes bundled with the Jupyter and Spyder development environments as well as the additional modules we will use in the course.

You are free to use a development environment of your choice on your own computer, as there are many good (possibly better) development environments.

If you choose a different development environment, you still should be familiar with Spyder and Jupyter for use in the labs.

Outline

Problem Solving

Program Development and Programming Languages

The Python Programming Language

Problem Solving

The purpose of writing a program is to solve a problem

Solving a problem consists of multiple activities:

- Understand the problem
- Design a solution
- Consider alternatives and refine the solution
- Implement the solution
- Test the solution

These activities are not purely linear – they overlap and interact

Algorithms

Steps used to solve a problem.

The steps in an algorithm should be precisely defined and the ordering of the steps is very important.

An algorithm defines:

- sequence of simple steps.
- flow of control process that specifies when each step is executed.
- a means of determining when to stop.

Algorithms are language independent; one solution can be implemented in any programming language.

Techniques used to develop algorithms (flowcharts, pseudocode).

Flowcharts and Pseudocode

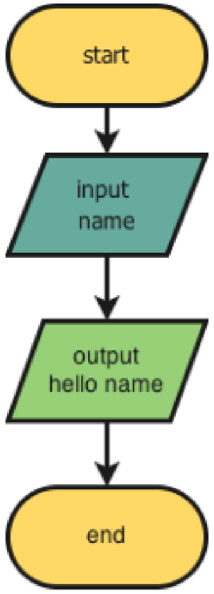
Example: A program that asks the user their name then says ‘Hello [Name]’:

Pseudo code:
an ordered version of your code written for human understanding rather than machine understanding.

```
1. Input name

2. Output ‘hello [name]’
```

Flowchart: a diagrammatic representation of a solution



Outline

Problem Solving

➡ Program Development and Programming Languages

The Python Programming Language

Computer Programs

The mechanics of developing a program include several activities:

- writing the program in a specific programming language (such as Python)
- translating the program into a form that the computer can execute
- investigating and fixing various types of errors that can occur

Software tools can be used to help with all parts of this process.

Spyder/Jupyter provide tools for all steps in program development.

Programming Languages

There are hundreds of programming languages, no best or worst language.

Some languages are better or worse for different applications.

Each programming language has:

- Set of primitive constructs(‘words’): numbers, strings, operators(+-/)
- A syntax (describes the order of the words)
- Semantics (defines the meaning of the sentences)

Syntax and Semantics

The *syntax rules* of a language define how we can put together symbols, reserved words, and identifiers to make a valid program

The *semantics* of a program statement define what that statement means (its purpose or role in a program)

A program that is syntactically correct is not necessarily logically (semantically) correct

A program will always do what we tell it to do, not what we <u>meant</u> to tell it to do

Errors in Programs

Syntactic errors

- common and easily caught

Semantic errors

- can cause unpredictable behavior
- some languages like Java check for these before running program, some such as Python do less checking.
- code has different meaning than what programmer intended
- program crashes, stops running
- program runs forever
- program gives an answer but different than expected

Outline

Problem Solving

Program Development and Programming Languages

➡ The Python Programming Language

Introduction to Python

General purpose programming language.

Can be used effectively to build almost any type of program.

Advantages:

- Relatively simple language.
- Easy to learn.
- Large number of freely available libraries.
- Easy to download and install.
- Runs under all operating systems.

Disadvantages:

- Because of minimal static semantic checking, not optimal for:
  - programs that have high reliability constraints (air traffic control systems, medical device).
  - large team projects, or extended length projects. (“too many cooks spoil the soup”)

Introduction to Python

Python is an interpreted language versus a compiled language.

In interpreted languages, the sequence of instructions (source code) is executed directly whereas in compiled languages the source code is first converted into a sequence of machine code.

Both have advantages and disadvantages.

Python Programs

A program (script) is a sequence of definitions and commands

- definitions evaluated
- commands executed by Python interpreter

Commands (statements) instruct interpreter to do something.

Commands can be typed directly in a shell or stored in a file that is read and evaluated by the interpreter.

A new shell is created whenever execution of a program begins.

Usually a window is associated with the shell.

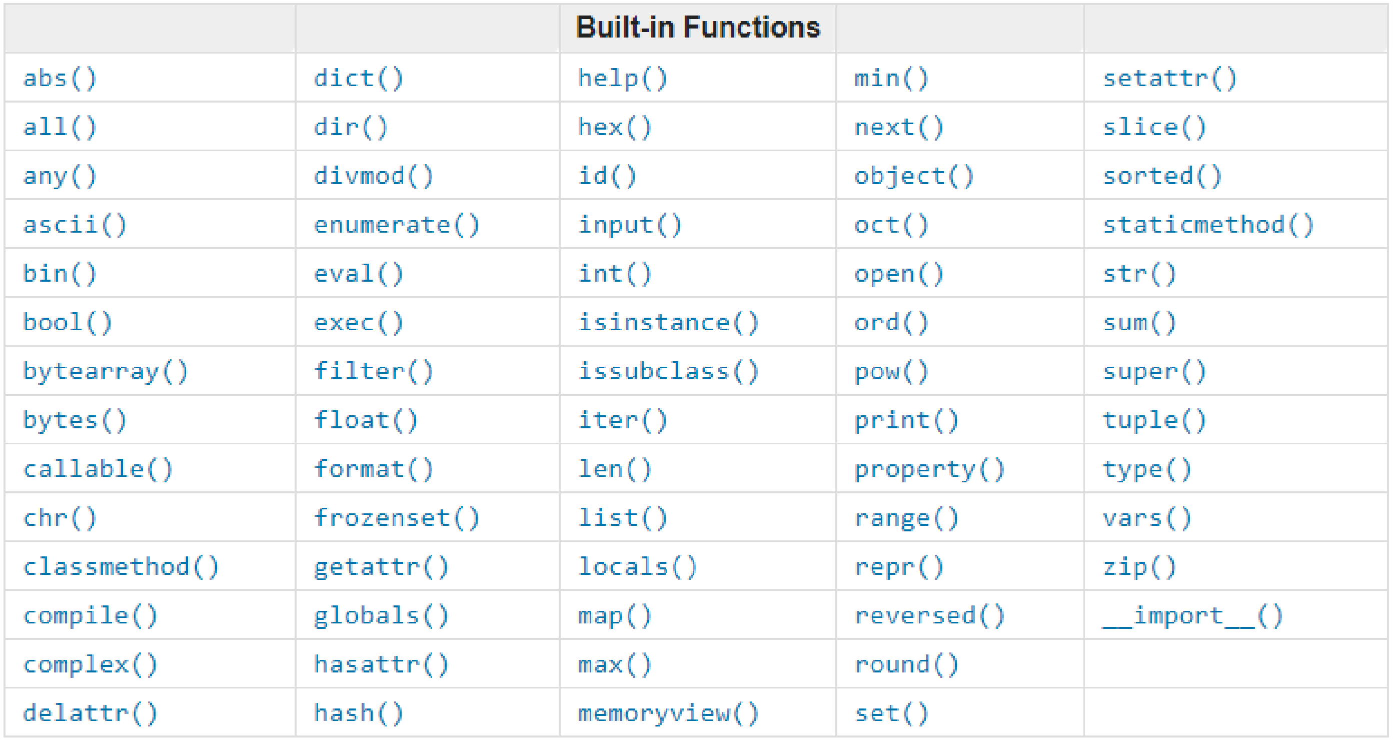
Shell vs. Script

We can print `‘Hello World’` in two different ways, using the shell (command line) or using a script(python file).

The `print()` command prints text to the screen, so the command `print(‘Hello World’)` will print the given text to the console.

Built-In Functions

The Python interpreter has a number of functions and types built into it that are always available.



		Built-in Functions		
abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

Complete list of built-in functions.

Objects

Programs manipulate data objects.

Objects have a type that defines the kinds of things programs can do to them.

- The value 5 can be multiplied, added, etc.
- ‘Hello world’ is a string that can be searched, sliced.
- Ana is a Student so she can register for courses, calculate her GPA, etc.
- Milo is an Employee, so he can get promoted, resign from his job, etc.

Objects are:

- scalar (cannot be subdivided)
- non-scalar (have internal structure that can be accessed)

Scalar Objects

`int` – represent integers, ex. 5

`float` – represent real numbers, ex. 3.27

`bool` – represent Boolean values True and False

`NoneType` – special and has one value, None

Can use `type()` to see the type of an object:

- `type("hello world")`
  ```
       str
  ```
- `type(5)`
  ```
       int
  ```
- `type(3.5)`
  ```
       float
  ```
- `type( True )`
  ```
      bool
  ```

Type Conversion

Sometimes we need to convert data from one type to another. For example, we may want to find the integer value of 3.2.

We can use built in functions to convert object of one type to another (type casting)

- `float(3)` converts integer 3 to float 3.0
- `int(3.9)` truncates float 3.9 to integer 3
- `bool(5)` converts non zero values to boolean True
- `bool(0)` converts zero value to boolean False

Variables

A *variable* is a name for a location in memory that references an object and allows us to associate labels with objects.

A variable has three things:

- a label : that is how we refer to it
- a type : what kind of a thing is stored in the box
- a value : what is in the box

Naming Variables

Variable names in Python must conform to the following rules:

- must start with a letter or underscore
- can only contain letters, numbers, and the underscore character _
- cannot contain spaces
- cannot include punctuation
- are not enclosed in quotes or brackets

The following names are valid variable names: `constant, new_variable, my2rules , SQUARES`

The following are invalid variable names: `a constant, 3newVariables, &sum`

Reserved Words

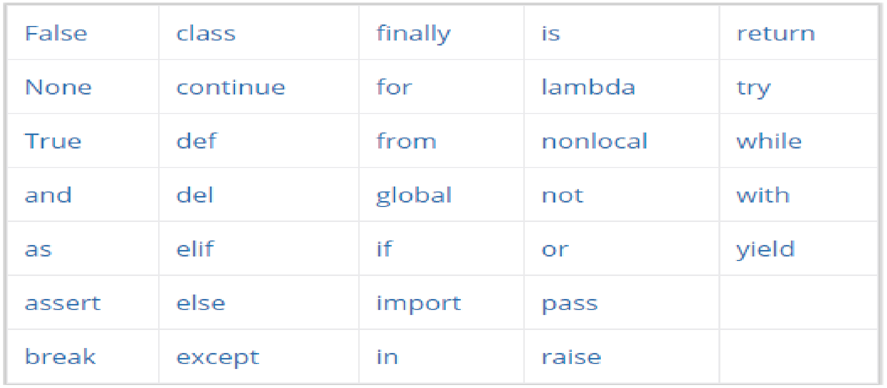
Reserved words are the keywords that have built-in meanings and cannot be used as variable names.

Each version of Python may have a different keyword list.

To see the list of reserved words in Python you can type the commands:

```
import keyword
   keyword.kwlist
```

Example list:



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

Binding Variables and Values (Assignment)

An assignment statement changes the value of a variable

The assignment operator is the = symbol.

Syntax of assignment statement:

$$\texttt{variableName} = \left\{\begin{array}{l}\text{value}\\ \text{variableName}\\ \text{expression}\end{array}\right.$$

Example:

```
total = 55
```

CHANGING BINDINGS

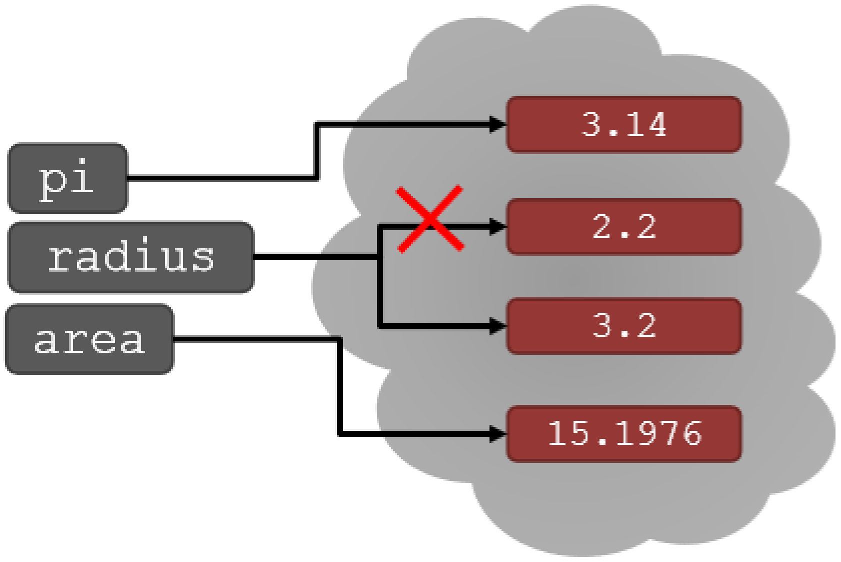
In python you can re-bind variable names using new assignment statements.

The previous value may still be stored in memory but the reference to the value no longer exists.

Value for area does not change until you tell the computer to do the calculation again.

Example:

```
pi = 3.14
radius = 2.2
area = pi*(radius**2)
radius = radius+1
```



Multiple Assignment

The expressions on the right side of an assignment are evaluated before any bindings are changed.

Example:

```
x, y = 2, 3
x, y = y, x
print('x=', x)
print('y=', y)
```

will print:

```
x = 3
y = 2
```

Expressions

Combine objects and operators to form expressions.

Expression has a value, which has a type.

Syntax for a simple expression:

```
<object> <operator> <object>
```

OPERATORS ON `ints` and `floats`

i+j -> the sum

i-j -> the difference

i*j -> the product

i/j -> floating point division

i//j -> integer division (value is truncated)

i%j -> the remainder when i is divided by j

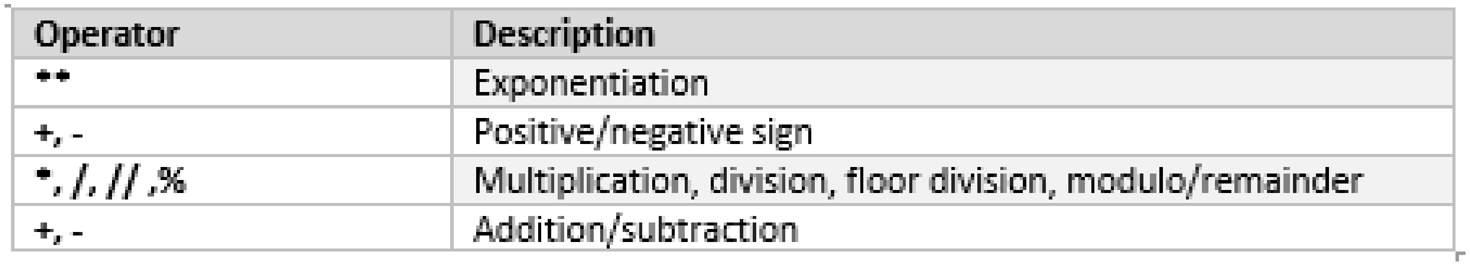
i**j -> i to the power of j

Python Arithmetic Operator Precedence

The order of evaluation can be changed by using parentheses to group sub-expressions.

For example: (x + y) * 2, first adds x and y then multiplies the product by 2.

The table below shows the arithmetic operator precedence.



Operator	Description
**	Exponentiation
+, -	Positive/negative sign
*, /, // ,%	Multiplication, division, floor division, modulo/remainder
+, -	Addition/subtraction

Quick Check

What are the results of the following expressions?

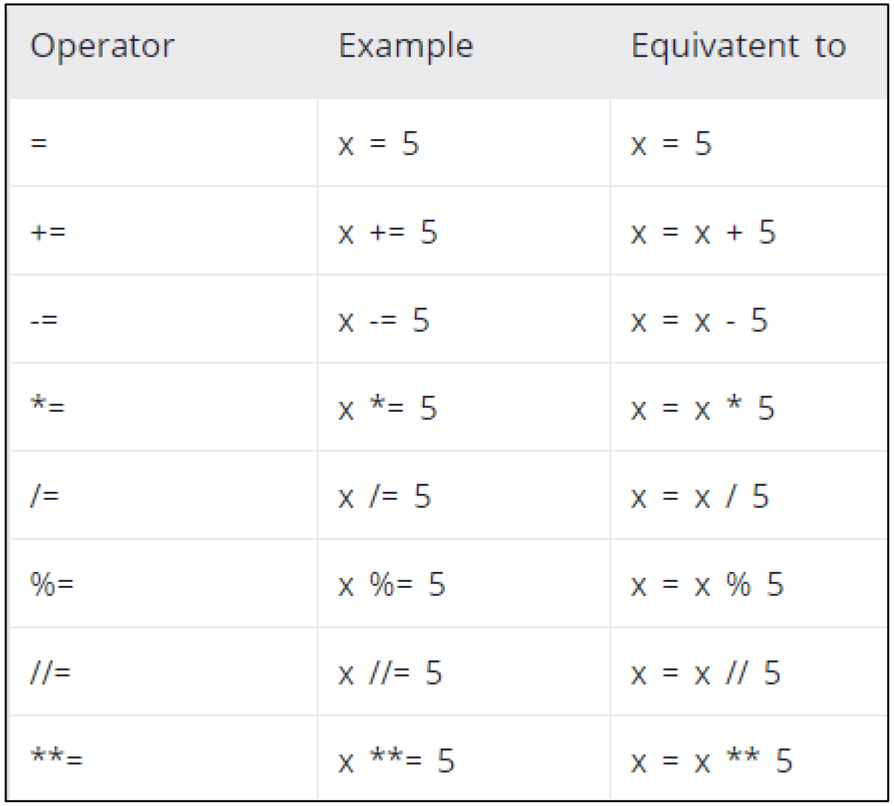
```
    12 / 5
   12 // 5
  10 / 4.0
 10 // 4.0
    4 / 10
   4 // 10
    12 % 3
    10 % 3
    3 % 10
```

Quick Check

What are the results of the following expressions?

```
  12 / 5  =  2.4
 12 // 5  =  2
 10 / 4.0 =  2.5
10 // 4.0 =  2.0
  4 / 10  =  0.4
 4 // 10  =  0
  12 % 3  =  0
  10 % 3  =  1
  3 % 10  =  3
```

Assignment Operators



Operator	Example	Equivatent to
=	x = 5	x = 5
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
%=	x %= 5	x = x % 5
//=	x //= 5	x = x // 5
**=	x **= 5	x = x ** 5

- See: `01_assignment.py`

Quick Check

In what order are the operators evaluated in the following expressions?

```
a + b + c + d + e          a + b * c - d / e
```

```
a / (b + c) - d % e
```

```
a / (b * (c + (d - e)))
```

Quick Check

In what order are the operators evaluated in the following expressions?

```
a + b + c + d + e
  1   2   3   4
```

```
a + b * c - d / e
  3   1   4   2
```

```
a / (b + c) - d % e
  2    1    4   3
```

```
a / (b * (c + (d - e)))
  4    3    2     1
```

Writing Good Code

Coding conventions are used to improve the readability of code and make it consistent across the wide spectrum of Python versions.

Provide guidelines about the code layout and style.

In this course, we will follow the PEP8 coding conventions.

Examples of conventions:

- Use blank lines to indicate logical sections.
- Use a single space before and after operators (exceptions exist)

See the PEP8 documentation for the complete list of rules.
Zen of Python: `import this`

Comments

Comments explain the purpose and process.

Comments are not interpreted and do not affect program execution.

Python comments can take two forms:

```
#single line comment

"""
multi-line comment, continues to end
symbol, across line breaks
"""
```

Comments should describe ‘why’ rather than how.

Multi-line comments are also called docstrings.

Docstrings provide documentation for functions, modules and classes which we will discuss later.

Formatting Strings: `f-string`

To format numeric or string values, use **`f-strings`**

Syntax:

```
   f'string to be formatted'
```

'string to be formatted' includes format fields, enclosed in curly braces {…}

The format fields are used to embed formatted values in the format string and include special characters that indicate how the values should be formatted (type of data and how it will be displayed).

Formatting Strings

```
salary = 7512.32165
print(f'Your salary is {salary}')
```

Output:

```
Your salary is 7512.32165
```

VS

```
salary = 7512.32165
print( f'Your salary is {salary:.2f}')
```

Output:

```
Your salary is 7512.32
```

Format Field: `{variable_index:format_type}`

```
print(f'Your salary is {salary:m.nf}')

m: places reserved for value, including decimal point.
n: number of decimal places
f: format specifier for float values


print(f'Your age is {age:md}')
m: places reserved for value
d: format specifier for int values

print(f‘Your name is {name:ms}')
m: places reserved for value
s: format specifier for string values
```

Formatting Example

```
salary = 7510.2685
age = 32
name = 'Jane'

print(f'Name: {name:10s} Age: {age:05d} Salary: {salary:.2f}')
```

Output:

```
Name: Jane       Age: 00032 Salary: 7510.27

print(f'Salary and age of {name:s} are {salary:12.3f} {age:5d} ')
```

Output:

```
Salary and age of Jane are     7510.269    32
```

Exercises

1. Write a program that converts 400°K to Celsius and displays the formatted result.
2. Write a program that finds the area and perimeter of a circle with the radius 1.5 cm and display the result. You can assume `pi` is the constant value 3.14.
3. Write a program that finds the sum of the first and last digit of some 4-digit number.
4. Write a program to calculate the number of hours and minutes needed for a car to travel 500 kilometers at the velocity 110 km/h.

See: `01_exercises.py or 01_exercises.ipynb`

Terms of Use