Python Fundamentals I

(basics, selection, repetition, functions, modules)

CPEN333 - 2023 W1

University of British Columbia

Electrical and Computer Engineering

Introduction

- Python is a popular, powerful, high-level, modern general-purpose programming language.
  - It is a widely used language for developing software and websites, task automation, data analysis, machine learning, …
- Python syntax is very clean, though it is different from the C-based languages.
  - For example, as a fundamental difference in syntax, indentation is essential for creating a block of code.
- This set of slides serves as an intro to the Python basics.

Objectives

- Given your past courses in programming, you should be able to learn and use Python fast.

- By the end of this set of slides, you should be able to:
  - Describe the general structure of a python program
  - Declare and use variables
  - Design branching structures (if, if-else, nested if, elif)
  - Design repetition structures (for, while, while-else)
  - Design functions and modules
  - Use built-in functions

Python 3

- We only use Python 3 in this course.
  - Although python 2 is at EOL (end of life), you may run into legacy code online that is based on python 2.
  - So be aware of the differences between Python 2 and Python 3 (https://docs.python.org/3.0/whatsnew/3.0.html).
- Python 3 is not meant to be backward compatible.
- As a simple example, the `print` statement is used differently:

```
print "Hello world"        # Python 2: print statement

  versus

print("Hello world")       # Python 3: as a function
```

Installing Python

- Even if your system comes with python (such as macOS), you are better off installing python 3 as described below.

- There are a number of options for installing python, here we discussed two of them.
  - Installing python through Anaconda installation
    - Recommended installation method
    - See next slide
  - Installing python though *python.org* installation package
    - https://www.python.org/downloads/
    - *python.org* is the de facto resource for installation and documentation.
  - Using online python tools or platforms.
    - e.g. for quick checks …

Installing Anaconda

- Installation: https://docs.anaconda.com/anaconda/install/index.html
  - The above link provides detailed instructions for installing anaconda on macOS, windows, Linux, …
  - It is the de facto installation for data science and machine learning development.

- Installing Anaconda will also come with a few very useful and related tools such as:
  - Navigator, Jupyter notebook, Spyder ...
  - its own package manager and many libraries

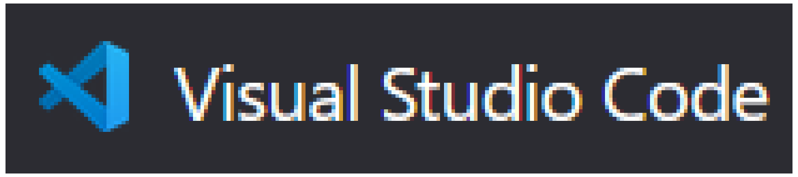
- The default version of python that comes with an up-to-date Anaconda is considered the required version of python used in this course.
  - Read any upcoming instructions carefully for any update to this rule.

IDE

- You have a number of options for the IDE.
  - Visual Studio Code (https://code.visualstudio.com/download)
    - a modern, versatile, very popular, and agile IDE
    - recommended IDE
  - IDLE
    - comes with python package downloaded from python.org
  - PyCharm (https://www.jetbrains.com/pycharm/download/)
    - a popular IDE from jetbrains
  - Spydr
    - comes with Anaconda
    - we may use it occasionally
  - …

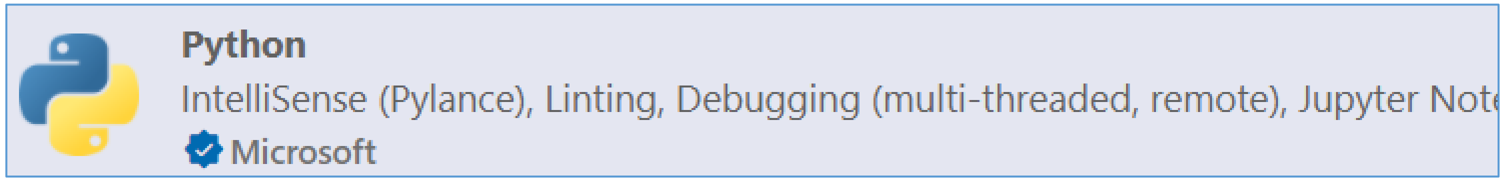
Installing VS Code

- Available for Windows, macOS, Linux, …
  - https://code.visualstudio.com/download



- It is lightweight yet very capable, with support for many languages
  - superbly extendable with many extensions available

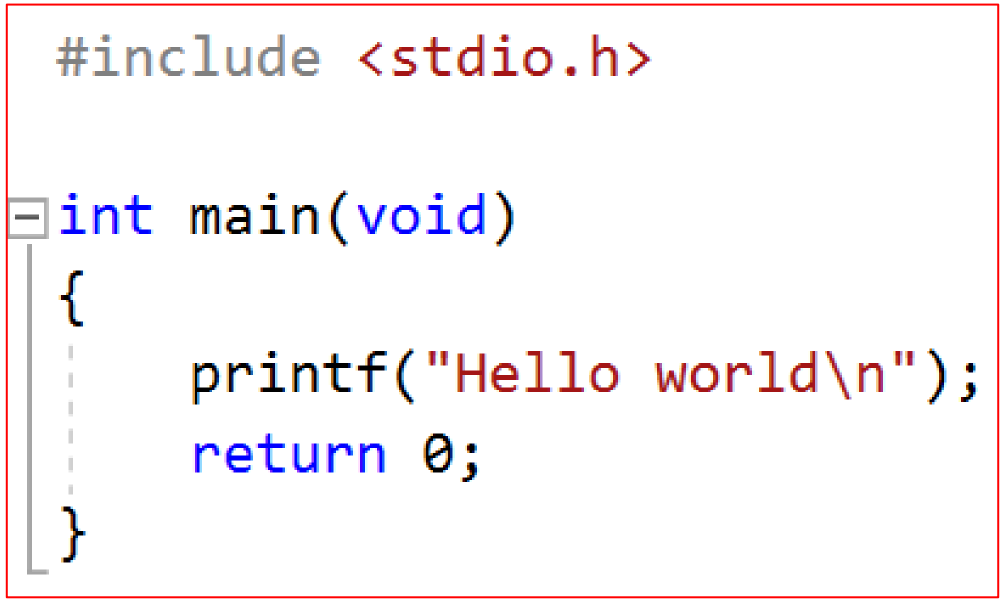
- Make sure to at least install the following extension:
  - Python extension for Visual Studio Code, from Microsoft


Python
IntelliSense (Pylance), Linting, Debugging (multi-threaded, remote), Jupyter Not
Microsoft

Hello World

- Python is classified as an interpreted language,
  - though there is the intermediary bytecode, and there are tools and implementations that compile it to other languages (e.g. C++).
- The infamous hello world program is shown below for Python and alternatively in C.

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



```c
#include <stdio.h>

int main(void)
{
    printf("Hello world\n");
    return 0;
}
```

Note that in python we can use either of double-quote or single-quote for a string literal. So the following is correct too:

```python
print('Hello world')
```

Comment Statements

- A comment in Python starts with the number sign/hash character, #, and extends to the end of the line.
  - Examples:

```
# This is a comment
print("Hello world") # This is a second comment

num = 1  # and this is the third comment
         # ... and now a fourth!
message = "# This is not a comment because it's inside quotes."
```

- For multi-line comments, use # multiple times or use triple quotes or triple double quotes (*docstrings*). A *docstring* is usually used as a first statement in a function, class, module … as documentation comment.
  - Example:

```
""" This function periodically controls the
    LEDs based on the required pattern."""
```

Variables

- A variable is created when it is first assigned a value to it.
- Python is case-sensitive.

```
>>> x = 1
>>> X = "Hello"
>>> print(x, X)
1 Hello
```

- Python has many typical types you may expect from a programming language built-in into the interpreter.
  - numerics (int, float), text (str), sequence (list, tuple, range), class, function, …
  - https://docs.python.org/3/library/stdtypes.html
- Python does not require the type of the variables to be specified before use.
  - The type will be inferred from the usage.

```
grade = 1
text = "Excellent"
threshold = 1.5
```

Variable Scope

- A tricky aspect of variable use in Python is careful attention to their scope.
- See the provided Jupiter Notebook file for examples.

Type system

- Python is dynamically typed, that is:
  - the interpreter type checks at runtime, and
  - the type of a variable is allowed to change over its lifetime.
- However, mechanisms such as type hints helps readability and allow us to have better type checking (by checkers, IDEs, linters, etc).
  - https://docs.python.org/3/library/typing.html

```
grade: int = 1
text: str = "Excellent"
threshold: float = 1.5
```

- Note that the Python runtime may not enforce function and variable type annotations.
  - Still very useful for functions, classes, …

```
>>> grade: int = 1
>>> grade = "hello"
>>> type(grade)
<class 'str'>
```

Indentation

- In C-based languages, we use curly braces (that is , {}) to group lines of code.
- Python requires indentation to indicate a block of code.
  - The indentation within the block must be consistent.

```
for version in range(3):
    text = f"file{version}"
    print(text)
```

- Compare:

```
for _ in range(3):
    print("A")
    print("B")
```

⟺

```
for _ in range(3):
    print("A")
print("B") # NOT a part of for
```

_ is a general purpose throwaway variable here

Selection (if Statement)

- We can use the if statement for selection. Examples:

```
if num >= 0:
    print("num is not negative")
```

```
if num >= 0:
    print("num is not negative")
else:
    print("num is negative")
```

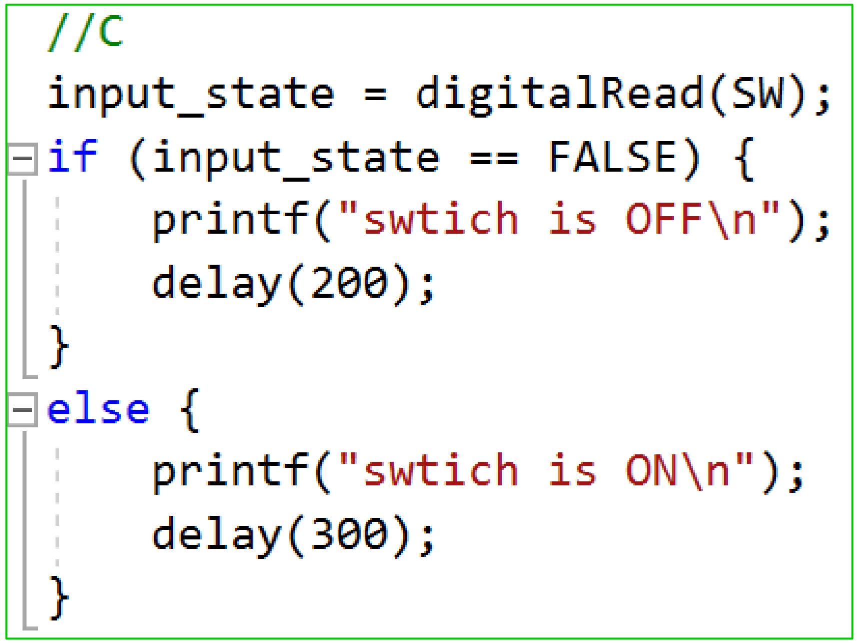
```
if num > 0:
    print("num is positive")
elif num < 0:
    print("num is negative")
else:
    print("num is zero")
```

- Note:
  - the indentation for the body of the *if* or *else*
    - This is mandatory and must be consistent
  - the colon at the end
  - that there is no need for parenthesis for the condition (optional).
  - `elif` is `else if`
- As usual, we can nest if statements.

if statement (cont.)

- Comparing an if-else statement in Python and C:

```python
#Python
input_state = GPIO.input(18)
if input_state == False:
    print('switch is OFF')
    time.sleep(0.2)
else:
    print('switch is ON')
    time.sleep(0.3)
```

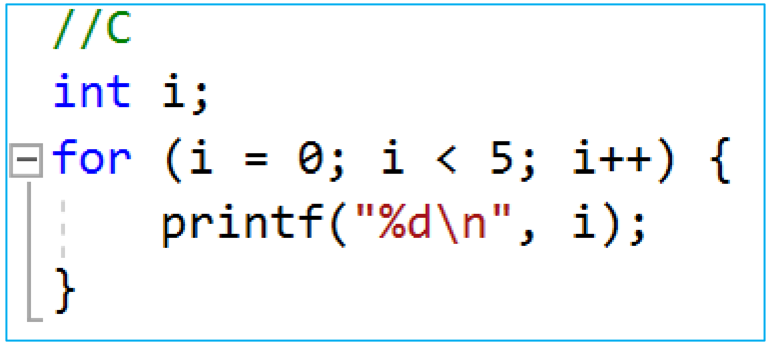


```c
//C
input_state = digitalRead(SW);
if (input_state == FALSE) {
    printf("swtich is OFF\n");
    delay(200);
}
else {
    printf("swtich is ON\n");
    delay(300);
}
```

Loops

- Here is a *for* loop statement in Python (and equivalently in C):

```
#Python
for i in range(5):
    print(i)
```



```
//C
int i;
for (i = 0; i < 5; i++) {
    printf("%d\n", i);
}
```

- Note:
  - the indentation for the body of the loop
  - the colon at the end
  - the use of range() for iterating.
- The *while* loop is also straightforward:

```
while True: #superloop
    led.on()
    sleep(1)
    led.off()
    sleep(1)
```

`True` is a python keyword and a built-in Boolean literal. (and so is `False`)

Functions

- Using functions in Python is quite straightforward.
  - We use the *def* keyword to define/create a function/method.
  - Here are a few examples in Python (and somewhat equivalently in C):

```
def func1():
    print("example 1")
```

```
//C
void func1(void)
{
    printf("example 1\n");
}
```

```
def func2(a, b):
    print(a,"+",b,"=",a + b)
```

```
//C
void func2(double a, double b)
{
    printf("%f+%f=%f\n",a,b,a+b);
}
```

```
def func3(a, b):
    return a + b
```

```
//C
double func3(double a, double b)
{
    return a + b;
}
```

Functions (cont.)

- It is very useful to use type hints in our functions.
  - In Python, every function returns something. If there are no return statement, then it returns None. None is a keyword, representing absence of a value (somewhat similar to null, nil, undef). Note that None is an object.

```python
def func1() -> None:
    print("example 1")
```

```python
def func2(a: float, b: float) -> None:
    print(a,"+",b,"=",a + b)
```

```python
def func3(a: float, b: float) -> float:
    return a + b
```

```python
# Examples of calling
#   these functions
funct1()

func2(1.1, 2)

func3(1, 2.1)
```

Python operators

- Arithmetic operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
★ **	Exponentiation
★ //	Floor division

similar to integer division in C, but here the operands can also be float

- Comparison operators

Operator	Description
==	Equal
!=	Not equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal

Python operators (cont.)

- Logical operators

Operator	Description	Example
and	Logical and	x > 1 and x < 10
or	Logical or	x < -1 or x > 1
not	Logical not	not (x > 4)

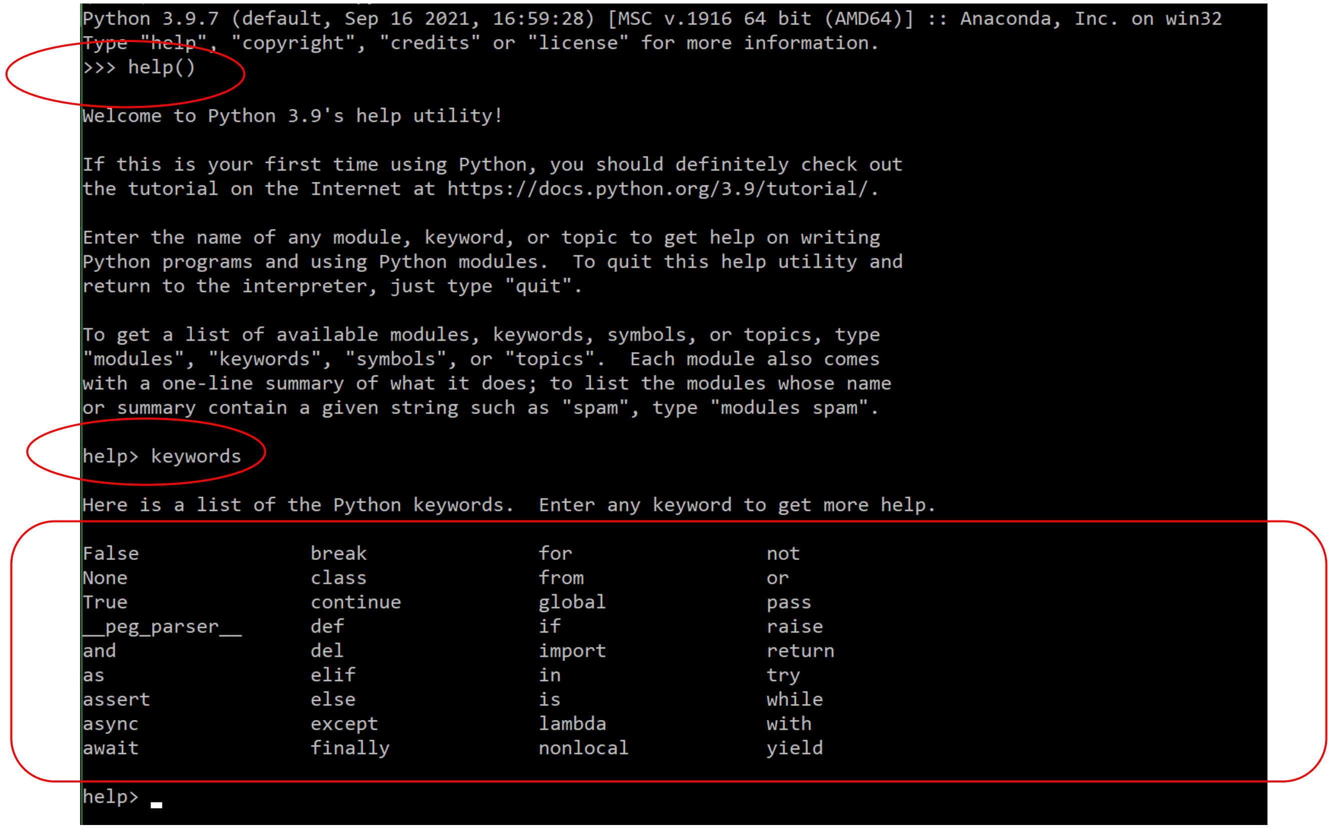
  - Do not confuse the logical operators with the bitwise operators (&, |, ~, ^, <<, >>).
- Assignment operator: =
  - Python has also the typical shorthand assignment operators: +=, *=, …
- Identity operator and membership operator

Operator	Description	Example
is	Identity operator (returns True if both are the same object)	x is y
in	Membership operator (returns True if the specified value is present in (a member of) the object)	x in y

See: https://www.w3schools.com/python/python_operators.asp

Keywords

- A simple way to see the list of python’s keywords is to use help():
  - https://www.python.org/dev/peps/pep-0233/
  - Alternatively: https://www.w3schools.com/python/python_ref_keywords.asp



```
Python 3.9.7 (default, Sep 16 2021, 16:59:28) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> help()

Welcome to Python 3.9's help utility!

If this is your first time using Python, you should definitely check out
the tutorial on the Internet at https://docs.python.org/3.9/tutorial/.

Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules.  To quit this help utility and
return to the interpreter, just type "quit".

To get a list of available modules, keywords, symbols, or topics, type
"modules", "keywords", "symbols", or "topics".  Each module also comes
with a one-line summary of what it does; to list the modules whose name
or summary contain a given string such as "spam", type "modules spam".

help> keywords

Here is a list of the Python keywords.  Enter any keyword to get more help.

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

help>
```

Built-in functions and types

- The python interpreter has a number of built-in functions and types.
  - https://docs.python.org/3/library/functions.html
  - https://docs.python.org/3/library/stdtypes.html
- Examples:
  - `print()`
  - `min(), max(), abs() and pow()`
  - `len()`
  - `complex (a complex number class)`
  - `dict (a dictionary class)`
  - `exec()`
  - `float, int, hex (types)`
  - …

Modules

- In Python, we can put the related definitions (functions, classes, variables) into a file called module.
  - The python standard library includes many useful modules: math, tkinter, threading, os, socket, … (https://docs.python.org/3/library/)
  - We use the import statement to import a module into our Python program.

```
import math
print(math.sqrt(math.sin(math.pi + 1) ** 2))
```

  - There are also many useful additional libraries: numpy, pytorch, openCV, …
- We can create our own module by storing our own function or class definitions in a file (with extension .py).
  - Similarly we use import to our module into any Python program.
  - The module must be placed in the same directory as the python program file.

import versus from-import

- By using `import tkinter` , you are directly importing the entire module (or resource/package).
  - Now we can use any of the functions … defined in tkinter.
  - For example to use its Tk methods:

```
root = tkinter.Tk()
```

- But `from tkinter import Tk` , only imports Tk from the module.
  - Now you can use Tk, without the need for stating “tkinter.”
  - For example:

```
root = Tk()
```

- Note that the binding between the namespaces (of the code and the imported module) is different.

`if __name__ == "__main__": ...`

- We normally use an if statement as show below:

```
def f1():
    ...
def f2():
    ...
if __name__ == "__main__":
    ...
```

- Simply put, it says that if this file is the one being run directly (as opposed to being imported), then run the statements in the if body.
  - **`__name__`** (two underscores preceding and following) is an internal variable that holds the name of the current module.
  - **`_main__`** is the name of the top-level environment of a program (https://docs.python.org/3/library/__main__.html), but when a python module is imported, **`__name__`** is set to the module’s name.
  - This allows a program to be directly run or be safely imported.

Examples

- There are alternative ways of writing the code:

```python
# imports here
def func1():
    print("funct1")
def func2():
    print("funct2")

if __name__ == "__main__":
    func1()
    func2()
```

```python
# imports here
def func1():
    print("funct1")
def func2():
    print("funct2")
def main():
    func1()
    func2()

if __name__ == "__main__":
    main()
```

Alternative (cont.)

- We could even code as follows, but it would be problematic if the code is imported (or example for unit testing, …)

```python
# imports here
def func1():
    print("funct1")
def func2():
    print("funct2")

func1()
func2()
```

```python
# imports here
def func1():
    print("funct1")
def func2():
    print("funct2")
def main():
    func1()
    func2()

main()
```

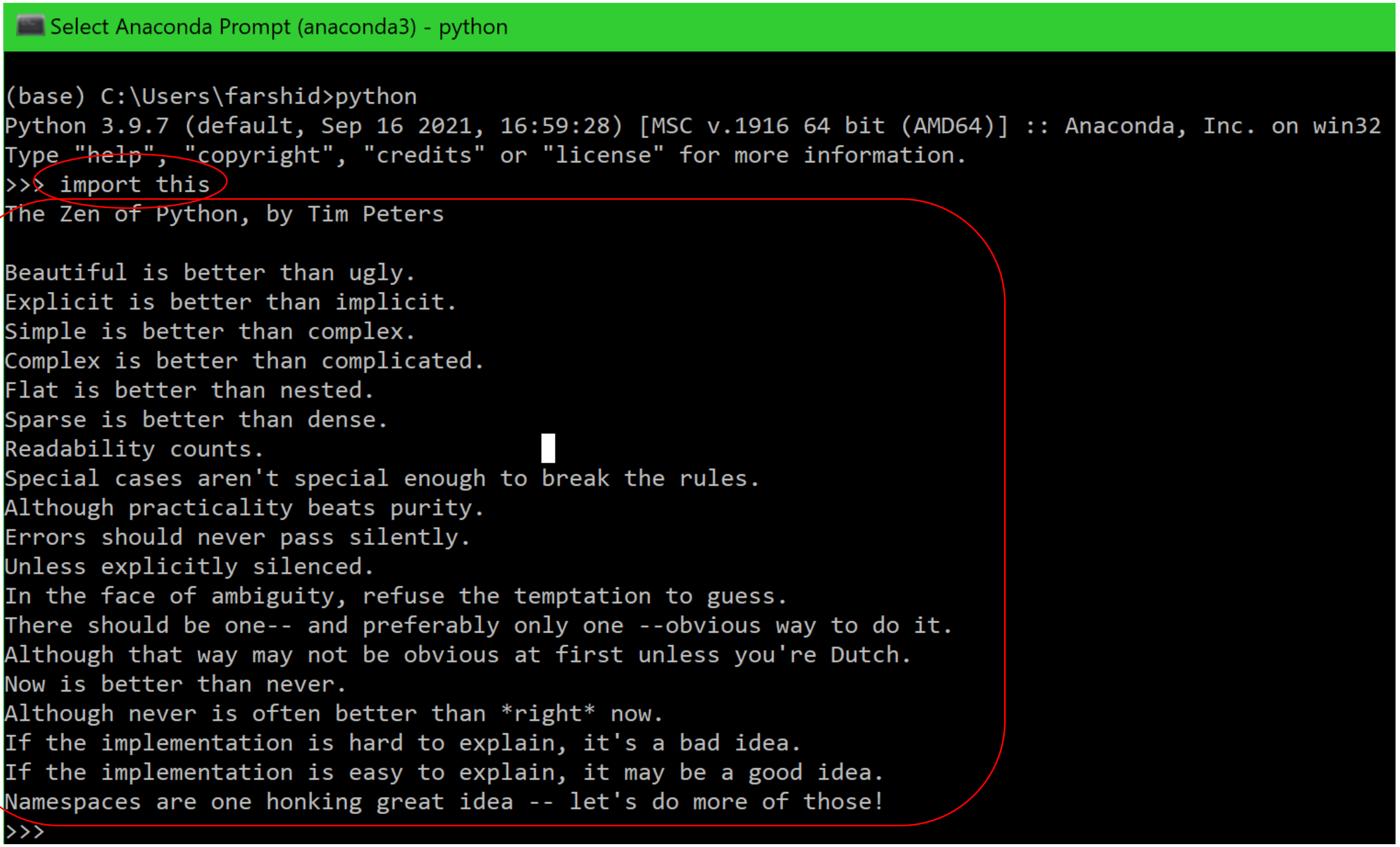
or even this of course,
if our code is simpler

```python
print("funct1")
print("funct2")
```

- The above alternatives is easily extended to python code with classes (object-oriented).

The Zen of Python

- Use import this:



```
(base) C:\Users\farshid>python
Python 3.9.7 (default, Sep 16 2021, 16:59:28) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
>>>
```

Style Guide

- There is an official style guide for python code (PEP 8):
  - https://www.python.org/dev/peps/pep-0008/

- We will try to follow the style as much as possible.

- When PEP does not make a recommendation, we may choose our own convention.
  - Example: using mostly double-quotes for string literal (for consistency with other languages we use like C, C#).

- A note on semicolon in python: in python we can use a semi-colon to denote separation (as opposed to C where it is statement termination).
  - So the following code is valid (but usually not a good-style):

```
print("Hello", end=' '); print("world!")
```

Lists

- A list can store a collection of data of any size.
  - You can access any list element by using the index operator, [].
  - You can use list methods to manipulate lists.
  - Lists are mutable.
  - Example:

```
list1 = [] # create an empty list
list2 = [2,3,4] # create a list with elements 2, 3, 4
list3 = ["red","green"] # create a list with strings

list4 = [2,"three",4] # A list can contain mixed types

two = list4[0] # stores 2 (element 0 of list4 in two)
list2.append(5) # modifies list2 to have 2, 3, 4, 5
```

- We will also discuss related such data types in more details soon.

References

- There is a great amount of good information on Python on the Internet.
  - Python documentation: https://docs.python.org/3/
  - Standard library: https://docs.python.org/3/library/
  - https://www.w3schools.com/python/
  - …
- There are also many good introductory books, e.g.:
  - Daniel Liang's ***Intro to Programming using Python***
- No worries, as you progress through the course and practice, you should master the language.