functions, input & output, importing modules.

Week 1 | Lecture 2 (1.2)

Today’s Content

Lecture 2.1

- The Programming Process

Lecture 2.2

- **Functions, input & output, importing modules**
- **Chapters 3**

What is a function?

- A function is a piece of code that you can "call" repeatedly to do one thing.
- Think about the **sin** key on your calculator. It takes in an angle, does some calculations and returns the sine of that angle.
- Python has **built-in functions** (today), but programmers can also create their own **user-defined functions** (next lecture).

easy
STAPLES

Why do we write functions?

- Let's consider our sine function.
- In Python, this could take 10 or more line of code to compute.
- If you have to compute the sine of an angle multiple times in your code, this means you have to repeat the same 10 lines of code over and over and OVER again!
- This is both inefficient and it creates more opportunities to bugs (mistakes) to creep into your code.

Open your notebook

Click Link:
1. Why do we write functions?

Why do we write functions?

- **Reuse:**
  - The practice of using the same piece of code in multiple applications.
- **Abstraction:**
  - A technique for managing the complexity of the code (how much do we really need to know?).
  - **model.fit(X, y)** → This could train a deep neural network.
- **Collaboration:**
  - Easy to read, Easy to modify, Easy to maintain.

- **#cleancode**

Calling Functions

- The general form of a function call:

```
function_name(arguments)
```

- Terminology
  - *argument*: a value given to a function.
  - *pass*: to provide an argument to a function.
  - *call*: ask Python to execute a function (by name).
  - *return*: give a value back to where the function was called from.

In **Python** names of variables and functions use low case and underscores.

```
function_name
Function_Name
FunctionName
```

Calling Functions

- The general form of a function call:

```
function_name(arguments)
```

- Terminology
  - *argument*: a value given to a function.
  - *pass*: to provide an argument to a function.
  - *call*: ask Python to execute a function (by name).
  - *return*: give a value back to where the function was called from.

Open your notebook

Click Link:
2. Function Call

Back to evaluation and expressions

- Last week we learned about the assignment statement (=).
- Remember, the value of the expression on the right-hand side (RHS) of the = sign is figured out first and then assigned to the variable on the left-hand side.
- This also applies if the thing on the RHS is a function!

First, the function is *called* while passing it an *argument*.

```
x = abs(-20)
```

Then, what the function *returns* is assigned to *x*.

Back to evaluation and expressions

- Last week we learned about the assignment statement (=).
- Remember, the value of the expression on the right-hand side (RHS) of the = sign is figured out first and then assigned to the variable on the left-hand side.
- This also applies if the thing on the RHS is a function!

Open your notebook

Click Link:
3. Back to Evaluation and Expressions

Breakout Session 1

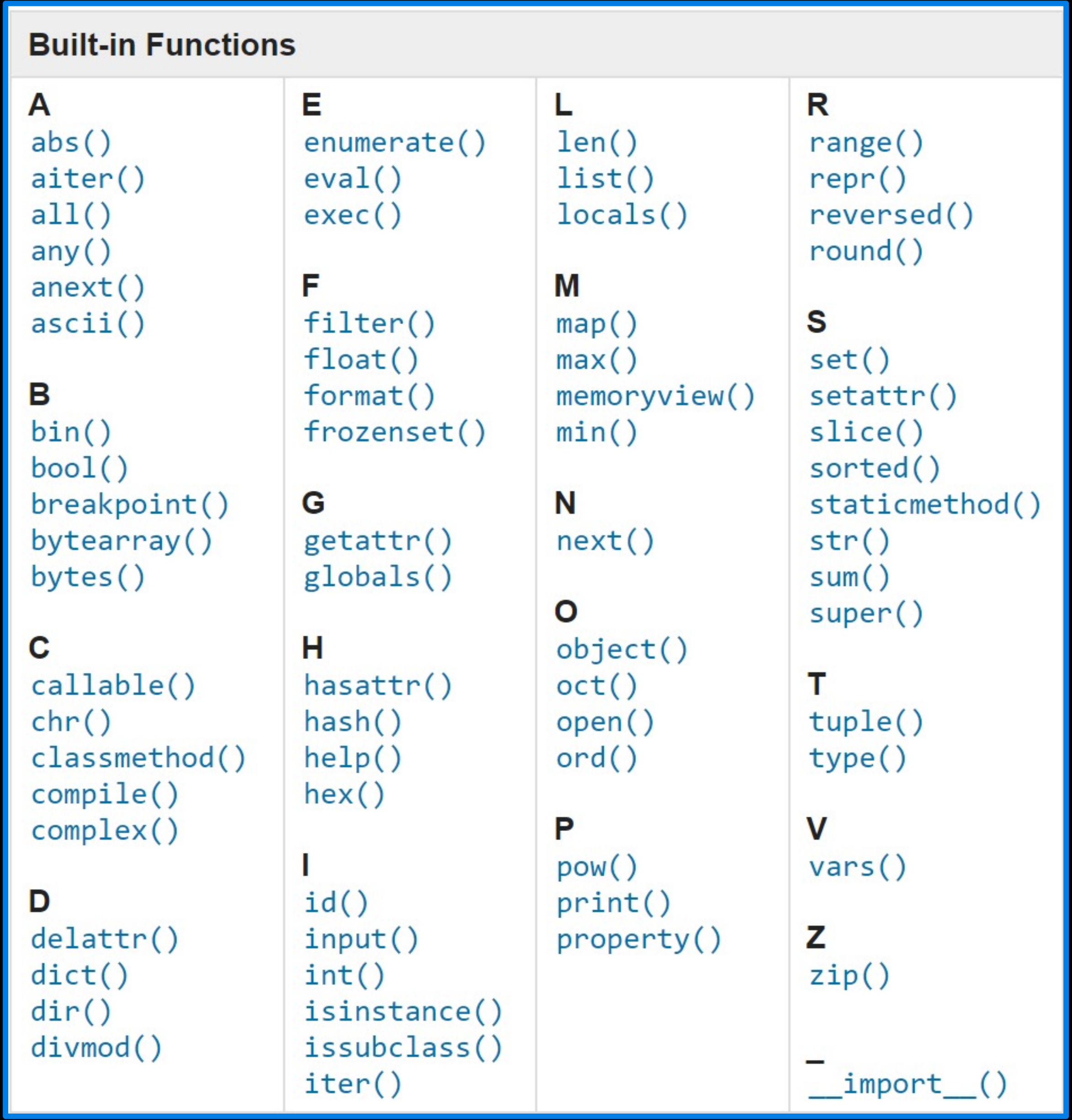
- $x = \frac{|y + z| + |y * z|}{y^\alpha}$
- where,
  - $y = -20$
  - $z = -100$
  - $\alpha = 2$
- What is x ?

Open your notebook

Click Link:
4. Breakout Session 1

Built-in Functions

- The *function_name* is the name of the function (like sin or print).
- Python has many built-in functions. Learn more about them here.



Built-in Functions

A
abs()
aiter()
all()
any()
anext()
ascii()

B
bin()
bool()
breakpoint()
bytearray()
bytes()

C
callable()
chr()
classmethod()
compile()
complex()

D
delattr()
dict()
dir()
divmod()

E
enumerate()
eval()
exec()

F
filter()
float()
format()
frozenset()

G
getattr()
globals()

H
hasattr()
hash()
help()
hex()

I
id()
input()
int()
isinstance()
issubclass()
iter()

L
len()
list()
locals()

M
map()
max()
memoryview()
min()

N
next()

O
object()
oct()
open()
ord()

P
pow()
print()
property()

R
range()
repr()
reversed()
round()

S
set()
setattr()
slice()
sorted()
staticmethod()
str()
sum()
super()

T
tuple()
type()

V
vars()

Z
zip()

_
__import__()

Built-in Functions

- The *function_name* is the name of the function (like sin or print).
- Python has many built-in functions. Learn more about them here.

Open your notebook

Click Link:
5. Built-in Functions

Function Help

- To get information about a particular function, call **help** and pass the function as the argument.
- **help** is one of Python's built-in functions.

- `help(abs)`

Open your notebook

Click Link:
6. Built-in Functions

Output

- Python has a built-in function named **print** for displaying messages to the user.
- The general form of a **print** function call:

```
print(arguments)
```

- The arguments can be of type `int`, `float`, `strings` and others we will discuss next week.

Open your notebook

Click Link:
7. Output

Input

- Python has a built-in function named **input** for reading inputs from the user.
- The general form of an **input** function call:

```
input(argument)
```

- The argument is the text you want displayed to the user.
  - *"What is your name?"*
- The value returned by the **input** function is always a string.

Open your notebook

Click Link:
8. Input

Breakout Session 2

- Write code to print out the following text:

```
"Hello, my name is {} and
I'm hoping to get a grade of
{} in APS106 this term."
```

- Where you see curly brackets {} you need to use the **input** function to prompt the user to enter that information.

Open your notebook

Click Link:
9. Breakout Session 2

Importing Functions and Modules

- Not all useful functions are built-in and they must be imported.
- Groups of functions are stored in separate Python files, which are called **modules**.
- Some modules come pre-installed with Python and other need to be installed separately.
  - For example, there are a lot of machine learning methods implemented in the scikit-learn modules.
- To get access to the functions in a module, you need to **import** the module.

Importing Functions and Modules

- The general for of an import statement is:
  - `import module_name`
- To access a function within a module:
  - `module_name.function_name`

- The dot is an operator:
  1. Look up the object that the variable to the left of the dot refers to.
  2. In that object, find the name that occurs to the right of the dot.

```
import math
math.sqrt(16)
```

Importing Functions and Modules

- The general for of an import statement is:
  - `import module_name`
- To access a function within a module:
  - `module_name.function_name`

I HAVE NO IDEA HOW PYTHON IMPORTS WORK
AND AT THIS POINT I'M TOO AFRAID TO ASK

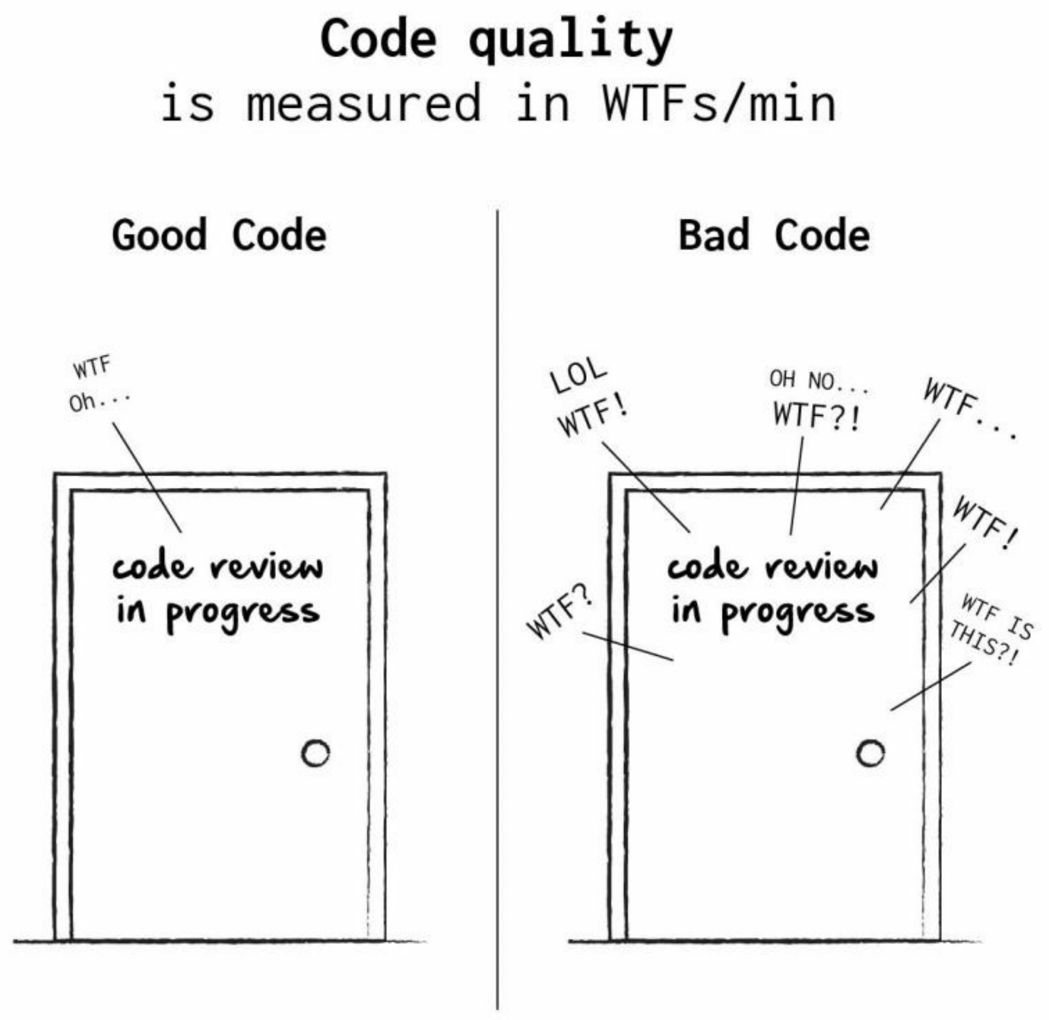
Open your notebook

Click Link:

10. Importing Function and Modules

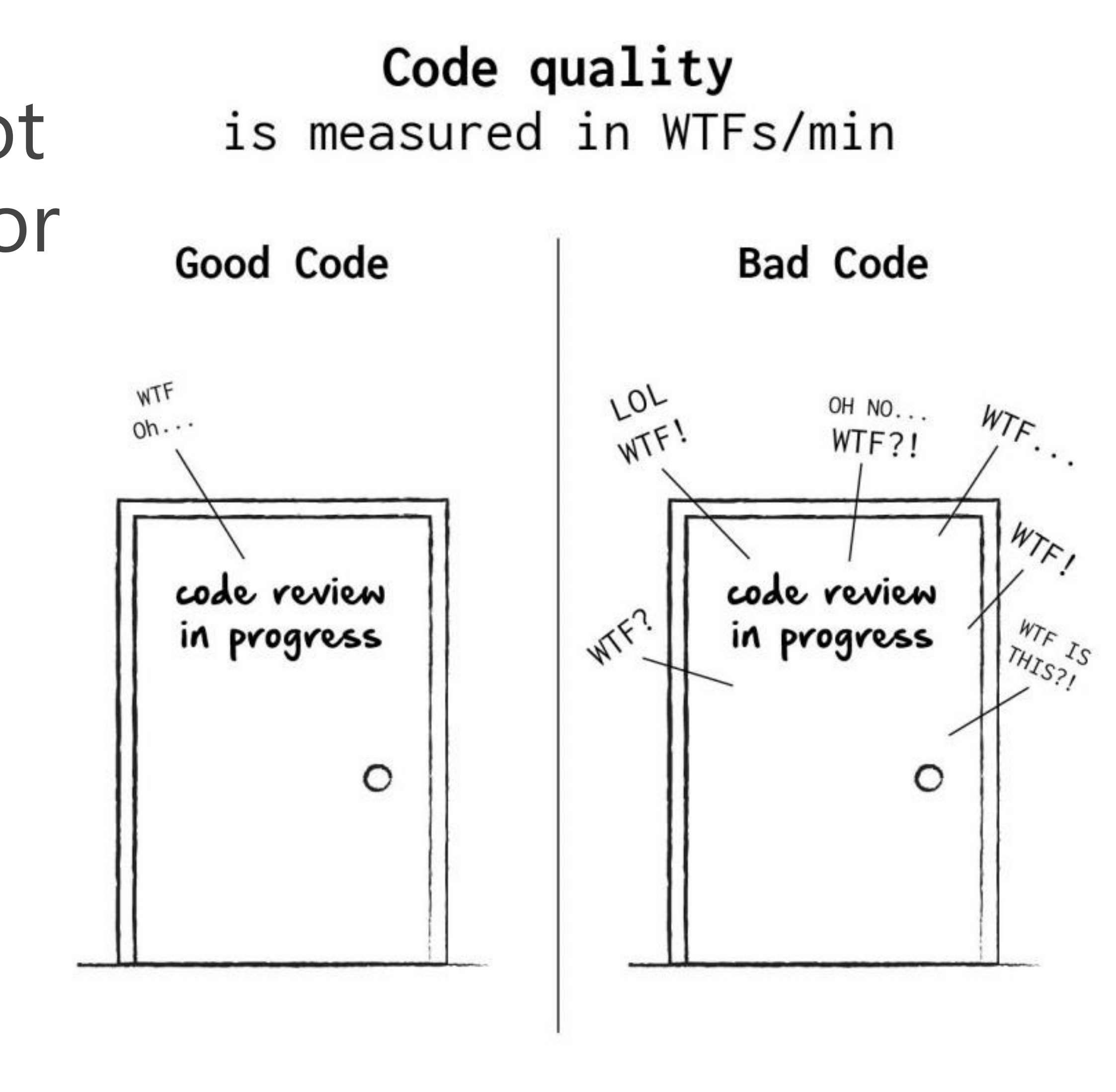
Defining Your Own Functions

- The real power of functions is in defining your own.
- Good programs typically consist of many small functions that call each other.
- If you have a function that does **only one thing** (like calculate the sine of an angle), it is likely not too large.
- If its not too large, it will be easy to test and maintain.



Defining Your Own Functions

- As a general rule, you should not write functions more than a 30 or 40 lines.
- Smaller is better: 10 or less is good.
- If you need something bigger, break it up into multiple functions.
- **#cleancode**



Function Definitions

- The general form of a function definition is:

```
def function_body(parameters):
    body
```

Colon

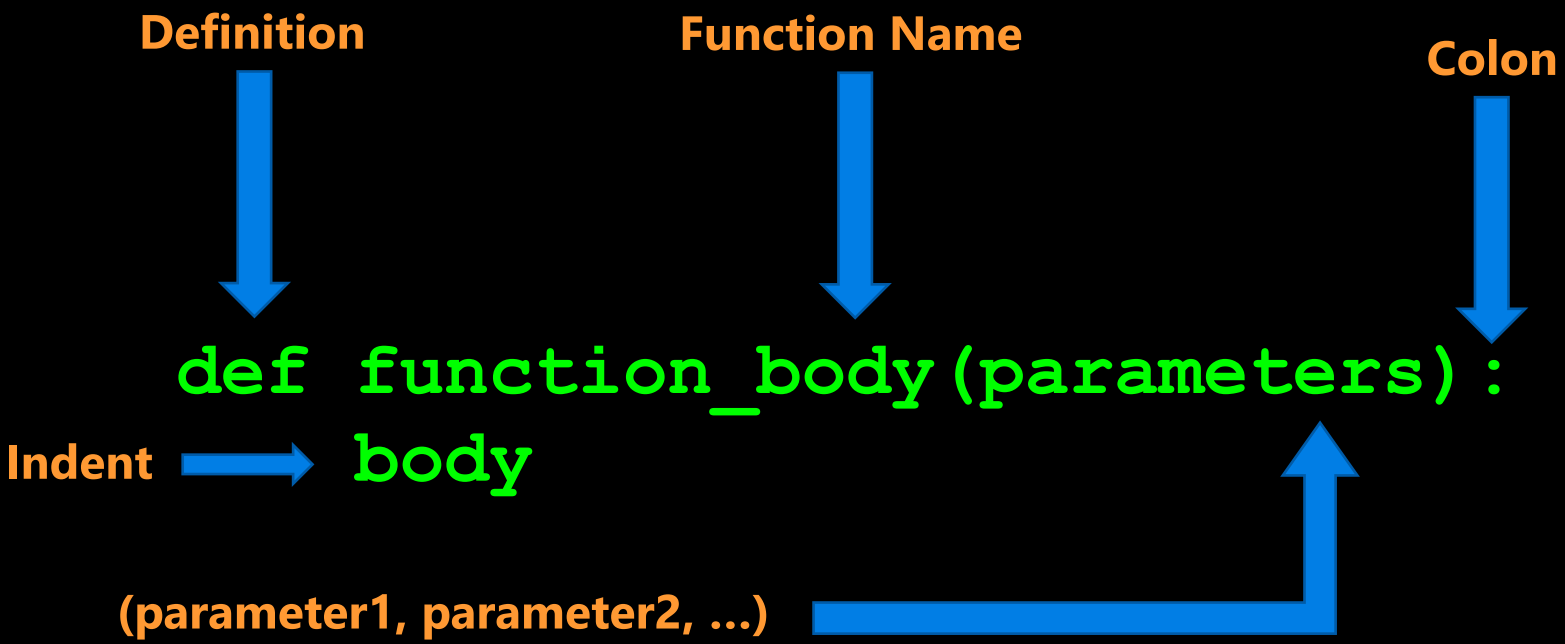
Indent

(parameter1, parameter2, ...)

- **def** is a keyword, standing for **definition**. All function definitions must begin with def. The def statement must end with a colon.
- **function_name** is the name you will use to call the function (like sin, abs but you need to create your own name).
- **parameters** are the variables that get values when you call the function. You can have 0 or more parameters, separated by commas. Must be in parenthesis.
- **body** is a sequence of commands like we've already seen (assignment, multiplication, function calls).
- **Important:** all the lines of body must be indented. That is how Python knows that they are part of the function.

Function Definitions

- The general form of a function definition is:



```
def function_body(parameters):
    body
```

Open your notebook

Click Link:
11. Defining Your Own Functions

Lecture Recap

- Functions.
- Calling functions.
- Importing Modules.
- Writing your own functions.

- More on functions next lecture!

functions, input & output, importing modules.

Week 1 | Lecture 2 (1.2)