CL05:
Introduction to Functions

Functions

Let you generalize problems for different inputs

Allow you to take solutions you defined in one place of your program and reuse them in other places of your program file.. and even in other program files!

Help you *abstract away* from certain processes

Abstraction Example

- Ordering a pizza…
  - You order a large cheese pizza
  - You don’t need to think about how they make the crust, got the ingredients, how long they bake it for, etc.
- round(x)
  - You round 10.25 down to 10 by calling round(10.25)
  - You don’t think about line by line how the some program is making this rounding decision

Calling a Function

Function Call: expressions that result in (“return”) a specific type

Common expressions:

- “Making a function call”
- “Using a function”
- “Invoking a function”

Looks like function_name(<inputs>)

E.g. print(“Hello”) , round(10.25), etc.

Examples…

print()

round()

randint()

Defining Functions

A function definitions are sub-programs that define what happens when a function is called.

Can be:

- Built-in
- Imported in Libraries
- DIY - Define in your python file

Pause to practice:

Please do the LS on Gradescope!

Function Syntax

Syntax for Calling A Function

function_name(<inputs>)

Syntax for Calling A Function

function_name(<argument list>)

```
print(“hello”)
round(10.25)
randint(1,7)
randint(1,2+5)
```

Syntax for Defining A Function

```
def function_name(<parameter list>) -> <return type>:
    """Docstring describing function"""
    <what your function does>
```

function name

parameter list

return type

```
def my_max(number1: int, number2: int) -> int:
    """Returns the maximum value out two numbers"""
    if number1 >= number2:
        return number1
    else: #number1 < number2
        return number2
```

signature

```
def my_max(number1: int, number2: int) -> int:
    """Returns the maximum value out two numbers"""
    if number1 >= number2:
        return number1
    else: #number1 < number2
        return number2
```

Call vs. Signature

Call (for calling a function):

```
function_name(<argument list>)
```

```
my_max(11, 3)
```

Signature (for defining a function) :

```
def function_name(<parameter list>) -> <return type>:
```

```
def my_max(number1: int, number2: int) -> int:
```

Call vs. Signature

Call (for calling a function):

variable_name: <return type> = function_name(<argument list>)

x: int = my_max(11, 3)

Signature (for defining a function) :

def function_name(<parameter list>) -> <return type>:

def my_max(number1: int, number2: int) -> int:

Call vs. Signature

```
x: int = my_max(11, 3)
```

```
def my_max(number1: int, number2: int) -> int:
```

Call vs. Signature

```
x: int = my_max(11, 3)
```

```
def my_max(number1: int, number2: int) -> int:
```

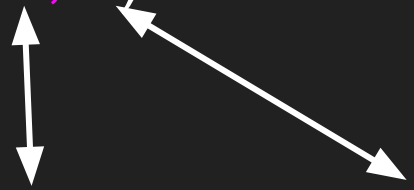
Call vs. Signature

```
x: int = my_max(11, 3)
```

```
def my_max(number1: int, number2: int) -> int:
```

Call vs. Signature

```
x: int = my_max(11, 3)
```



```
def my_max(number1: int, number2: int) -> int:
```

Call vs. Signature

```
x: int = my_max(11, 3)
```

“arguments”

```
def my_max(number1: int, number2: int) -> int:
```

“parameters”

Pause to practice:

Please do the LS on Gradescope!