



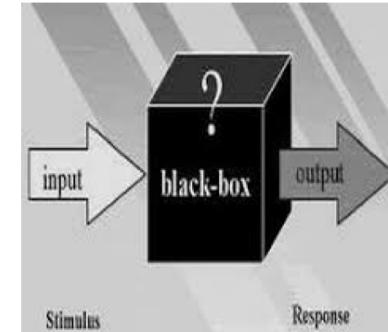
OMIS30

FUNCTIONS

week 4 class 2, fall 2019

What is a function?

- Think of a function as a machine that does a specific task
- The function 'machine' has a defined input and output
- You put something known in and get something known out



What is a function?

KNOWN INPUT -> PROCESS -> EXPECTED OUTPUT

function ()

- If you put the wrong thing in what will happen?





Definition

A function is an object that:

- 1.** Takes in pre-specified data as arguments
- 2.** Processes the data in pre-determined way
- 3.** Returns the data after processing

This way of programming minimizes
“unintended side effects”



Function syntax

Declare the function definition

```
def skill_to_expert(argument):
    """Takes the name of a skill and tells you what
    to call the expert at that skill

    USE EXAMPE : out=skill_to_expert('art')
    ARGUMENTS: string with skill name"""

    expert = argument + 'ist'

    return expert
```

The input is processed in the body of the function

NOTE the indent of 4 spaces or one tab to delimit the body of the function

Terminate the function def with a ":"

The Docstring gives information to the user and can be called with help ()

Return specifies the output

use "''' triple quotes



Make the function object

```
4 def skill_to_expert(argument):
5     """Takes the name of a skill and outputs expert name
6     USE EXAMPE : out=skill_to_expert('art')
7     ARGUMENTS: string with skill name"""
8
9     expert = argument + 'ist'
10    return expert
11
```

```
1 print(skill_to_expert)
```

```
<function skill_to_expert at 0x10be41ea0>
```

- Before we execute the function it is just an object (like a book or a cell phone)
- It is just sitting in object space doing nothing!
- When we print it Python tells us its type (function) and memory address



Start with a working example

```
4 out=skill_to_expert('art')
5 print(out)
```

artist

Since there is a good example in the documentation, we can start with that

Make sure everything is in working as expected



Moving forward

- We verified that the example works
- Try it with a different word
- Remove the variable assignment so it simply outputs to the workspace

```
7 skill_to_expert('BBQ')  
8
```

'BBQist'

** NOTE the output is not bound to any variable so we can see it but then it's GONE!

This is a good way to test rapidly your function but if you want to use it again bind it to a variable like we did with "out" (on the last slide)



Functions have expected inputs

```
4 skill_to_expert(1)

-----
TypeError                                     Traceback (most recent call last)
<ipython-input-11-2399ce3bc3fc> in <module>()
      2
      3
----> 4 skill_to_expert(1)

<ipython-input-6-ad7ada7d86cd> in skill_to_expert(argument)
      9     ARGUMENTS: string with skill name"""
     10
--> 11     expert = argument + 'ist'
     12
     13     return expert

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

- If you send the wrong variable type into a function, Python emits an error and tells you specifically what went wrong (read it from the bottom up)
 - We'll get to errors more later!



What is Functional Programming?

Functional Programming is a style of programming

- The main tenet of Functional Programming is that the programmer has complete control over what occurs in the program.
- The elimination of unintended **side effects**
- **Purity** of code
 - a pure function takes a **defined input** and returns a **defined output**



What is Functional Programming?

As shown in the example above, the function clearly does one thing:

```
skill_to_expert('skill') -> 'skillist'
```

```
skill_to_expert(1) -> TypeError!
```

- In other words, it avoids unintended side effects and does a defined job



Benefits of functional programming (discuss)

Modularity - it should be easy to remove and replace functions without affecting other code areas

Reusability - clear definition makes reuse in other context possible

Abstraction - the details of the function internals are obscured allowing the programmer to think about higher order code processes

Scalability - clear definition of task blocks allows replication and scaling

Ease of troubleshooting - broken code can be traced to single isolated functions



Learn about the function - Docstrings matter!

```
1 help(skill_to_expert)
```

```
Help on function skill_to_expert in module __main__:
```

```
skill_to_expert(argument)
```

```
Takes the name of a skill and tells you what  
to call the expert at that skill
```

```
USE EXAMPE : out=skill_to_expert('art')
```

```
ARGUMENTS: string with skill name
```

- We can call the help() function:
- Takes your function name as an argument and returns its docstring
- Hopefully there is a good doc string with a working example to get us started!



More Functions

```
1 import random
2 # random generates pseudo-random numbers from a given set
3
4 def coin_flip(tosses):
5     '''coin_flip takes an integer number of tosses
6     and outputs a list of random outcomes'''
7
8     outcome = []
9     coin = ['heads', 'tails']
10    for toss in range(tosses):
11        outcome.append(random.choice(coin))
12    return(outcome)
13
14 coin_flip(5)
```

What's the expected output?



More Functions

```
1 import random
2 # random generates pseudo-random numbers from a given set
3
4 def coin_flip(tosses):
5     '''coin_flip takes an integer number of tosses
6     and outputs a list of random outcomes'''
7
8     outcome = []
9     coin = ['heads','tails']
10    for toss in range(tosses):
11        outcome.append(random.choice(coin))
12    return(outcome)
13
14 coin_flip(5)
15
```

```
['tails', 'tails', 'heads', 'tails', 'heads']
```



String Format (str.format)

Python's str.format() method of the string class allows for variable substitutions and value formatting, allowing for elements to be joined into one string through positional formatting.

```
Print("Robert has {} rabbits.".format(300))
```



String Format (str.format)

Python's str.format() method of the string class allows for variable substitutions and value formatting, allowing for elements to be joined into one string through positional formatting.

```
Print("Robert has {} rabbits.".format(300))
```

Output: Robert has 300 rabbits.



String Format (str.format)

A variable can also be assigned to be equal to the value of a string that has formatter placeholders:

```
open_string = "Robert loves {}."  
Print(open_string.format("open source"))
```



String Format (str.format)

A variable can also be assigned to be equal to the value of a string that has formatter placeholders:

```
open_string = “Robert loves {}.”  
Print(open_string.format(“open source”))
```

Output: Robert loves open source.



String Format (str.format)

You can use multiple pairs of curly braces when using formatters, this allows the addition of another variable.

```
open_string = “Robert loves {} {}.”  
Print(open_string.format(“open source”, “software”))
```

Output: Robert loves open source software.