CDS 230
Modeling and Simulation I

Module 6

Functions

Dr. Hamdi Kavak
http://www.hamdikavak.com
hkavak@gmu.edu

GEORGE MASON UNIVERSITY

Center for Social Complexity

Functions

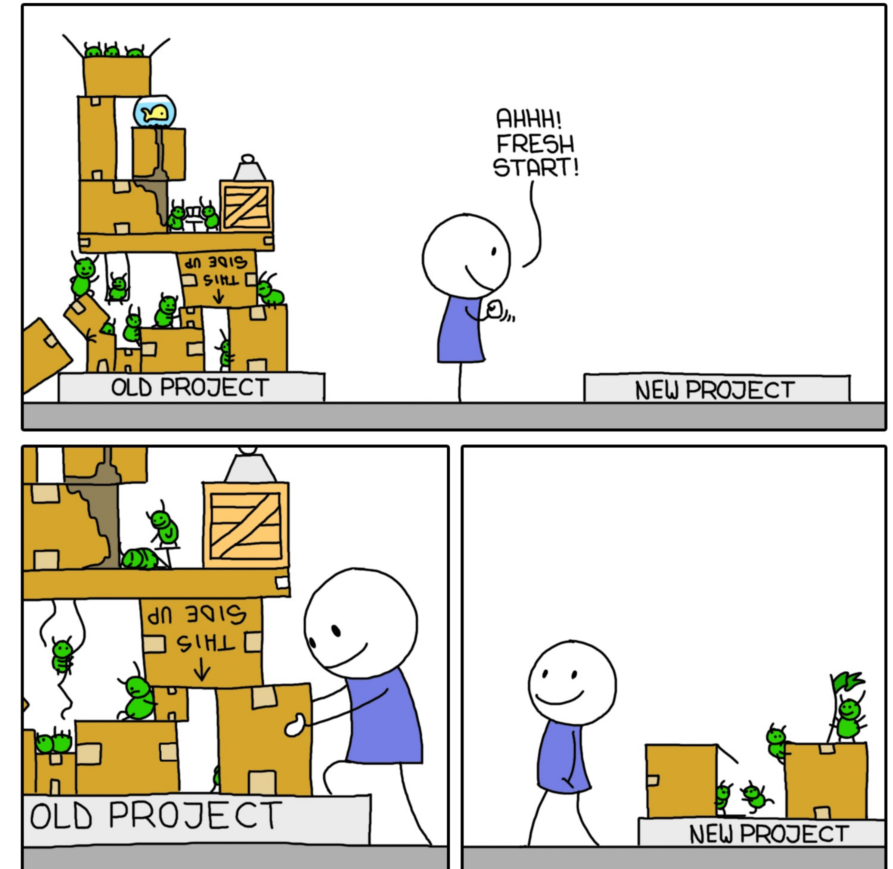
- A mechanism to group a set of statements together and call them using a given name.
- Designed for a specific task
- Actually, we have been using functions
    - `print(“Hello, world!”)`
    - `abs(-50)`
    - `list([1,2,3])`
    - `math.sin(0.0)`

GEORGE MASON UNIVERSITY

Center for Social Complexity

Why do we need functions?

- Code re-use
  - No need to re-invent the wheel
  - Makes coding more efficient
- Code organization
  - Separation of tasks
  - Improves readability
  - Easier to collaborate



Source: https://www.monkeyuser.com/2018/code-reuse/

Defining functions

Name of the function (given by you; case sensitive)

Keyword for defining a function

```
def square_root(num):
    s = num ** 0.5
    print(s)
```

Argument (parameter) to the function (not required).

Colon

Body of the function (indented)

Calling functions

- Calling signature (name and parameters) should match definition

```python
def square_root(num):
    s = num ** 0.5
    print(s)
```

Let's say we call this function with argument 4

```python
square_root(4)
```

- Should be introduced before first usage

```python
square_root(4)
def square_root(num):
    s = num ** 0.5
    print(s)
```

```
NameError: name 'square_root' is not defined
```

```python
def square_root(num):
    s = num ** 0.5
    print(s)

square_root(4)
```

```
2.0
```

`return` in functions

- `return` keyword is used to get value (s) out from functions
- you can use the returned value: `variable = function_name(param)`

Ways to return (or not return)

No return

```
def square_root(num):
    s = num ** 0.5
    print(s)
```

One value return

```
def square_root_r(num):
    s = num ** 0.5
    return s
```

Multiple values return? sort of..

```
def square_root_r2(num):
    s = num ** 0.5
    return -s,s
```

Let’s say we call each function with parameter value 16. Output?

GEORGE MASON UNIVERSITY
Center for Social Complexity

`print()` vs. `return()`

Printing and returning may seem similar, but there are important differences.

Print

- Used to tell *people* (us) what Python is doing.
- Good for figuring out issues when programming.
- Can be used anywhere we want.
- Can't save printed value in variables.

Return

- Used to tell *Python* what it itself is doing.
- Can only be used at the end of functions.
- Once Python hits a return, the function stops.
- Can save returned value in variables.
- Have to print to see returned value.

Arguments (passing values to functions)

- You can define as many you want and even call them unordered

```
def sum_4(a,b,c,d):
    s = a + b + c + d
    return s
```

```
sum_4(1,2,3,4)
```

10

```
sum_4(d=4, c=3, b=2, a=1)
```

10

- Optional arguments are possible

```
def convert_distance(distance, conversion="mile to km"):
    if conversion == "mile to km":
        return distance * 1.60934
    elif conversion == "km to mile":
        return distance * 0.621371
    else:
        print("Unsupported conversion is requested. Supported types are:")
        print("mile to km")
        print("km to mile")
```

Guess the output?

```
convert_distance(100)
```

```
convert_distance(100, conversion="km to mile")
```

```
convert_distance(100, conversion="mile to meter")
```

Variable scope

- Defined a variable within a function?
  - They are called local variables
  - It will be available to that function only
  - It will be live while function is being executed

```
def sum_4(a,b,c,d):
    s = a + b + c + d
    return s
```

local variable

Python's variable scope priorities

1. Local scope
2. Enclosing scope
   - for nested functions
3. Global scope
4. Built-ins

```
s = "This is a global variable"
def sum_4(a,b,c,d):
    g = "This is a local variable"

    # if we want to access the gloval s variable,
    #    we meed to use global keyword

    def sum_2(e,f):
        g = e + f
        # if we want to access enclosing g variable,
        #    we need to use nonlocal keyword

        return g

    s = sum_2(a,b) + sum_2(c,d)
    return s
```

Recursive functions

- A function that can call itself

```
def factorial(n):
    if n == 1:
        return 1
    return n * factorial(n-1)
```

- Alternative to iterative (loop etc.) coding but can be a bit slower
- Has the potential to shorten your code and make it look elegant
- Proper stop condition is needed, otherwise your code will crash

Example 1 – basic understanding

- Write a function named `sum_all` that takes a list and returns the sum of values.

```
values = [4,10,11,56]
sum_all(values)
```

```
81
```

Example 2 – Fibonacci sequence

- Fibonacci sequence: Each number is the sum of the two previous numbers.

F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}	F_{16}	F_{17}	F_{18}	F_{19}	F_{20}
0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610	987	1597	2584	4181	6765

You are given the first two numbers

Write a function that prints the above Fibonacci series based on F's index.

E.g.,: `fib(4)` will print: 0, 1, 1, 2, 3

So

- You should master on creating functions.
  - returns
  - *arguments* (values passed to functions)
  - *variable scope* (at least local vs. global variables)
- New coding assignments can and will ask you to write functions.
  - E.g.: write a function named `hello()` which takes no argument and prints "Hello, World!" as shown below.

```
hello()
```

```
Hello, World!
```