SKILLPILLS

# Skill Pill: Introduction to Programming

## Lecture 3: Functions

Vsevolod Nikulin

Okinawa Institute of Science and Technology
*vsevolod.nikulin@oist.jp*

March 26, 2019

OIST

# Overview

SKILLPILLS

1. Definition of a function
   - What is a Function?
   - Examples
   - Excercises
2. Side Effects
   - Side Effects Overview
   - Local vs Global Variables
   - Examples
3. High-Order Functions
   - Functional Variables
   - Functional Arguments and Returning Values
   - Examples
   - Excercise

# Definition of a function

# What is a function?

- A function is a self-sustained, reusable piece of code and is callable by name
- Functions encapsulate tasks
- Functions *may* have input argument(s) and *may* return output value(s)
- Functions in programming are not the same thing as in math! (side effects!)

# Simple Example

```
#Function Definition
def bad_pizza():
    """
    Prints a phrase
    """

    print("Pineapple Pizza bad")

#Function Call
bad_pizza()
```

# Simple Example: Argument

```
#Function Definition
def bad_pizza(ingredient_str):
    """
    Prints a phrase

    Keyword Argument:
        ingredient_str (string): Input word
    """

    print(ingredient_str + " Pizza bad")

#Function Call
bad_pizza("Anchovy")
```

# Simple Example: Argument and Output Value

SKILLPILLS

```
#Function Definition
def bad_pizza(ingredient_str):
    """
    Makes a phrase

    Keyword Argument:
        ingredient_str (string): Input word
    Returns:
        String: Output phrase
    """

    return ingredient_str + " Pizza bad"

#Function Call
phrase = bad_pizza("Anchovy")
print(phrase)
```

# Excercise 1

```
def my_line(my_number):
    """Prints a line consisting of my_number times "-".

    Keyword arguments:
        my_number (int) -- Number of "-"
    """
```

# Excercise 2

```
def my_sum(my_list):
    """Calculates the sum of a given list.
    Keyword arguments:
        my_list (list) -- Given list.
    Returns:
        Float -- Sum of given list.
     """

def my_mean(my_list):
    """Calculates the mean of a given list.
    Keyword arguments:
        my_list (list) -- Given list.
    Returns:
        Float -- Mean of given list.
     """
```

# Excercise 3

```
def my_triangle(my_a, my_b):
    """Calculates the area of a triangle given
    the length of the two shorter sides of the triangle.
    Keyword arguments:
        my_a (float) -- Length of side a.
        my_b (float) -- Length of side b.
    Returns:
        Float -- Area of the triangle.
    """

def my_circle(my_radius):
    """Calculates the area of a circle given its radius.
    Keyword arguments:
       my_radius (float) -- Radius of the circle.
    Returns:
       Float -- Area of the circle.
    """
```

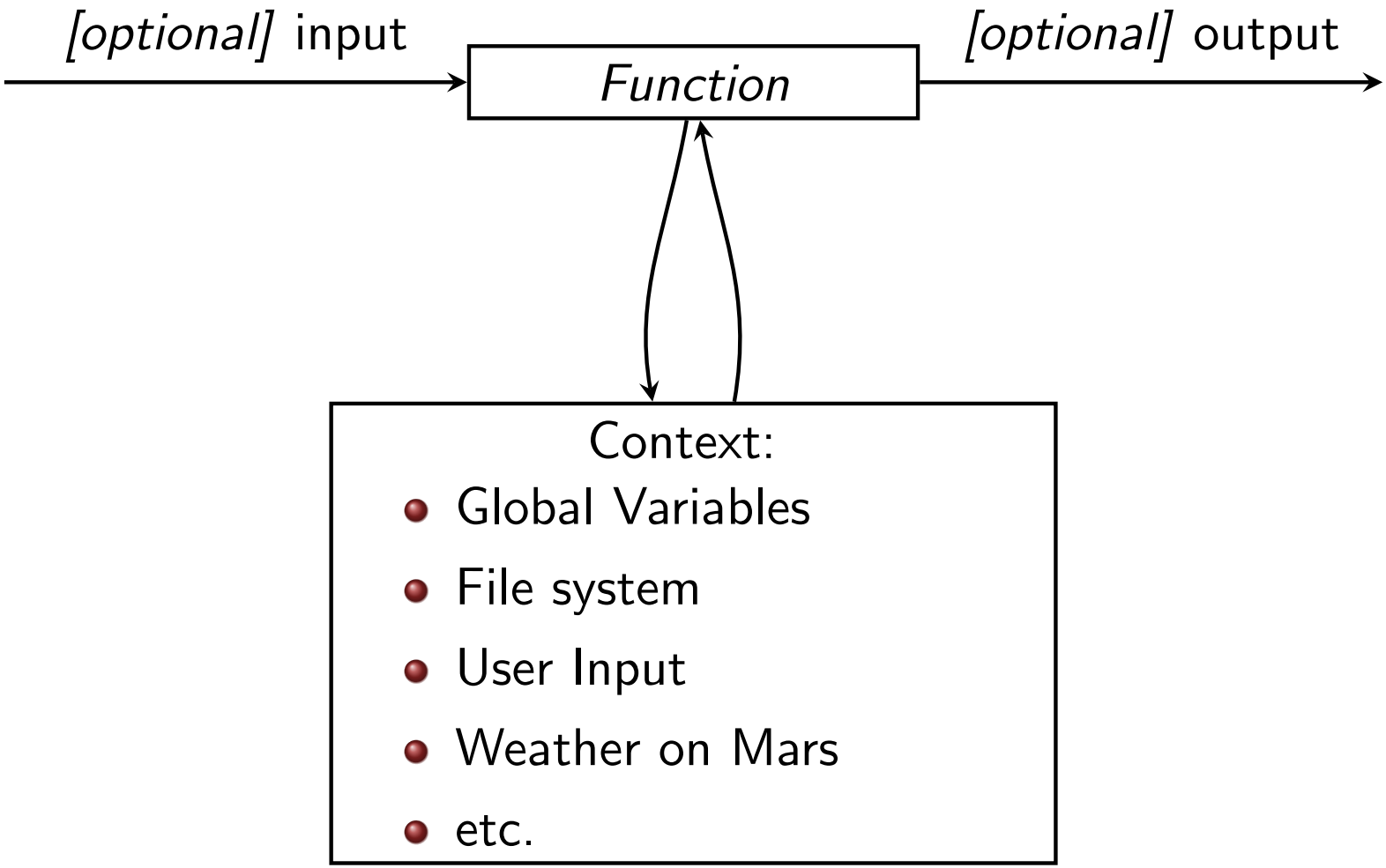
# Excercise 4

```
def my_reverse(my_string):
    """Reverses a given string.
    Keyword arguments:
        my_string (string) -- Given string.
    Returns:
        String -- Reversed string.
    """

def my_count(my_string, my_letter):
    """Counts the number of instances of a given letter in a given string.
    Keyword arguments:
        my_string (string) -- Given string.
        my_letter (char) -- Given letter.
    Returns:
        String -- Number of instances of given letter in given string.
     """
```

# Side Effects!

In general, we have



# Local Variables vs Global Variables

SKILLPILLS

```
global_phrase = "What Is the Airspeed Velocity"
                "of an Unladen Swallow?"
def bridge_keeper_questions(input_phrase):
    """
    Prints questions from the Keeper of the Bridge of Death
    """
    print("Stop. Who would cross the Bridge of Death "
          "must answer me these questions three, "
          "'ere the other side he see.")
    local_phrase = "What is you name?" #Local variable
    print(local_phrase)
    input_phrase = "What is your quest?" #What is happening here?
    print(input_phrase)
    global_phrase = "What is your favorite color?" #And here?
    print(global_phrase)
phrase = "Sir Lancelot"
bridge_keeper_questions(phrase)
print(global_phrase)
print(phrase)
```

# Changing a Global Variable Inside a Function

SKILLPILLS

```
global_phrase = "Sir Lancelot"

def switch_knight():
    """
    Changes global variable 'global_phrase'
    """
    global global_phrase

    global_phrase = "Sir Galahad"

print(global_phrase)
switch_knight()
print(global_phrase)
```

# Changing an Argument Variable Inside a Functio

SKILLPILLS

```
def recruit_knight(knights, new_knight_str):
    """
    Adds a knight to a party.

    Keyword Argument:
        knights (list of strings): Party of Knights
        new_knight_str (string): Name of a new Knight
    """

    knights.append(new_knight_str)

round_table_knights = ['Sir Arthur',
                       'Sir Lancelot',
                       'Sir Galahad']
recruit_knight(round_table_knights,  'Sir Bedevere')
print(round_table_knights)
```

# Global vs Local Variables Overview

SKILLPILLS

- There are three types of variable inside a function: **local**, **global** and **argument** variable
- Assigning a value to a **global** or **argument** variable inside a function will create a new *local* variable which overshadows it
- In order to change this behavior for **global** variables, you need to explicitly declare it using *global* keyword
- Arguments are passed *by reference*: you can modify these variables, but not by assigment.

# High-Order Functions

# Functional Variable

SKILLPILLS

You can assign functions as values for variables!

```
def add(x, y):
    return x + y

def sub(x, y):
    return x - y

f = add
print(f(2, 1))
f = sub
print(f(2, 1))
```

# High-order functions (1)

SKILLPILLS

You can pass functions as arguments to other functions!

```
def evaluate(f, x):
    """
    evaluates function f at point x
    keyword argument:
        f (function): function to evaluate
        x (any): value of the argument for function f
    returns:
        f(x)
    """
    return f(x)
```

# High-order functions (2)

SKILLPILLS

You can return functions from other functions!

```
def curry_add(x):
    """
    Makes a function which adds x to input
    keyword argument:
        x (number): value to add
    returns:
        function: function which adds x to input
    """
    def add(y):
        return x + y

    return add

add_5 = curry_add(5)
print(add_5(2))
print(curry_add(5)(2))
```

# Excercise 5

```
def make_censor_list_function(condition, censor):
    """
    Makes a function, which takes a list and
    returns a new list for which elements satisfying condition
    are replaced using function censor
    Keyword Argument:
        condition (function(a) -> boolean): condition function
        censor (function(a) -> a): censor function
    Returns:
        function([a]) -> [a]: resulting function
    """
```

# The End