Functions Part 1

Section 1
Chapter 4

Quiz 8

Functions

- Functions are like vending machines
  - Receive input
  - Process the input
  - Produce the output

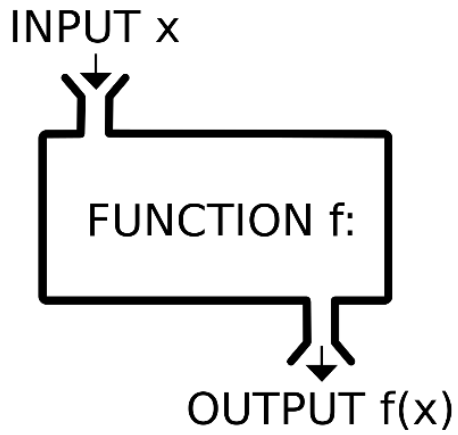

TABLE 4.1 **Some Python built-in functions.**

Function	Example	Input	Output
int	int(2.6) is 2	number	number
chr	chr(65) is 'A'	number	string
ord	ord('A') is 65	string	number
round	round(2.34, 1) is 2.3	number, number	number

Built-in Functions

- When the output of a function is a single value, the function is said to **return** its output

- Items inside parentheses are called arguments

```
>>> num = int(3.7)  # literal as an argument
>>>
>>> num1 = 2.6
>>> num2 = int(num1)  # variable as an argument
>>>
>>> num1 = 1.3
>>> num2 = int(2*num1)  # expression as an argument
```

```
>>> len("CityU", "HK")
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    len("CityU", "HK")
TypeError: len() takes exactly one argument (2 given)
>>>
>>> len(["CityU", "HK"])
2
>>> len("CityU HK")
8
```

User-defined Functions

- A user-defined function resembles the following:

```
def functionName(par1,  par2, ...):
    indented block of statements
    return  expression
```

  - **def** is short for define
  - The header must end with colon
  - Statements are in the indented block
  - The returned expression can be omitted

User-defined Functions

- Three ways to pass arguments to parameters:
  - **Pass by position**, pass by keyword, and pass by default value.

- In this section, we consider pass by position - arguments in calling statement matched to the parameters in function header based on order.

- Parameters and return statements are optional in function definitions

- Function names should describe the role performed

Example 1: area of circle

- Example 1: The area of a circle depends on its radius.

```
>>> area_circle(1)
3.14
>>> area_circle(2)
12.56
```

Example 2: extract first name

```
>>> # After running the script for the user-defined function.
>>> firstName("George Carlin")
'George'
```

Functions with Several Parameters

```
def pay(wage, hours):
    ## Calculate weekly pay with time-and-a-half for overtime.
    if hours <= 40:
        amount = wage * hours
    else:
        amount = (wage * 40) + ((1.5) * wage * (hours - 40))
    return amount
```

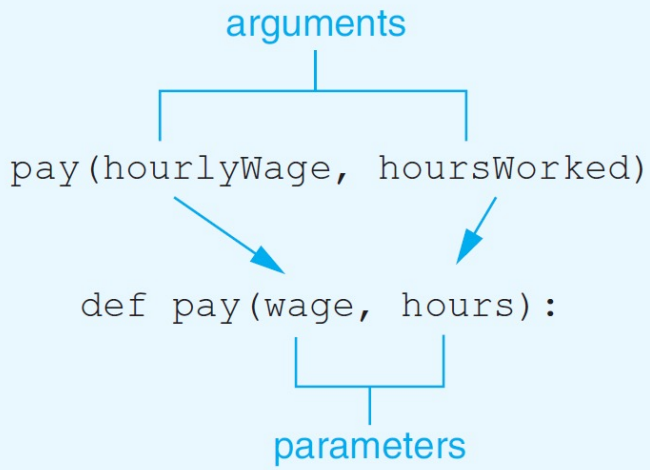

FIGURE 4.3 Passing arguments to a function.

Example 3: Future value

Suppose the function takes three arguments:

- Beginning balance
- Rate of return
- The number of periods.

```
>>> fV(1000, 0.03, 4)
1125.51
```

Example 4: Present value

```
>>> pV(1125.51, 0.03, 4)
1000.0
```

Boolean-valued Functions

- Boolean-valued functions return True or False

- Example 5: Recall Classwork 5 – a year is a leap year if it is divisible by 4 except when it is divisible by 100 and not by 400. Define a function that evaluates whether a year is a leap year or not.

```
>>> leap(2021)
False
>>> leap(2020)
True
>>> leap(2000)
True
>>> leap(1900)
False
```

Functions that do not Return Values

```
# Example 6: A function that does not return values

def hello(name):
    print("Hello", name)

# print is different from return.
# print() just display some text.
# return will stop the execution of the current function,
   # and assign an outcome value to the function.
```

Functions without Parameters

```
Enter a radius: 2
12.56
```

```
# Example 7: function without parameter

def main():
    radius = eval(input("Enter a radius: "))
    print(area_circle(radius))

def area_circle(radius):
    return 3.14*radius**2

main()

#note that the function main() has no parameter.
```

Scope of Variables

- Variable created inside a function can only be accessed by statements inside that function
  - Those variables cease to exist when the function is exited

- Variable is said to be local to function or to have local scope

- If two variables are created in two different functions have the same name, they have no relationship to each other.

Scope of Variables

- Scope of a variable is the portion of the program that can refer to it

- To make a variable global, place assignment statement that creates it at top of program.
  - Any function can read the value of a global variable
  - The value cannot be altered inside a function unless the altering statement is preceded by a statement of the form:

```
global globalVariableName
```

Scope of Variables

```
def trivial(x):
    y = x+1
    return y
```

```
>>> trivial(3)
4
>>> y
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    y
NameError: name 'y' is not defined
>>> x
Traceback (most recent call last):
  File "<pyshell#10>", line 1, in <module>
    x
NameError: name 'x' is not defined
```

Scope of Variables

```
def trivial(x):
    global y
    y = x+1
    return y
```

```
>>> trivial(3)
4
>>> y
4
>>> x
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    x
NameError: name 'x' is not defined
```

Passing a Value to a Function

- A program shows there is no change in the value of the argument

```
def triple(num):
    num = 3 * num
    return num

num = 2
print(triple(num))
print(num)
```

[Run]

```
6
2
```

Passing a Value to a Function

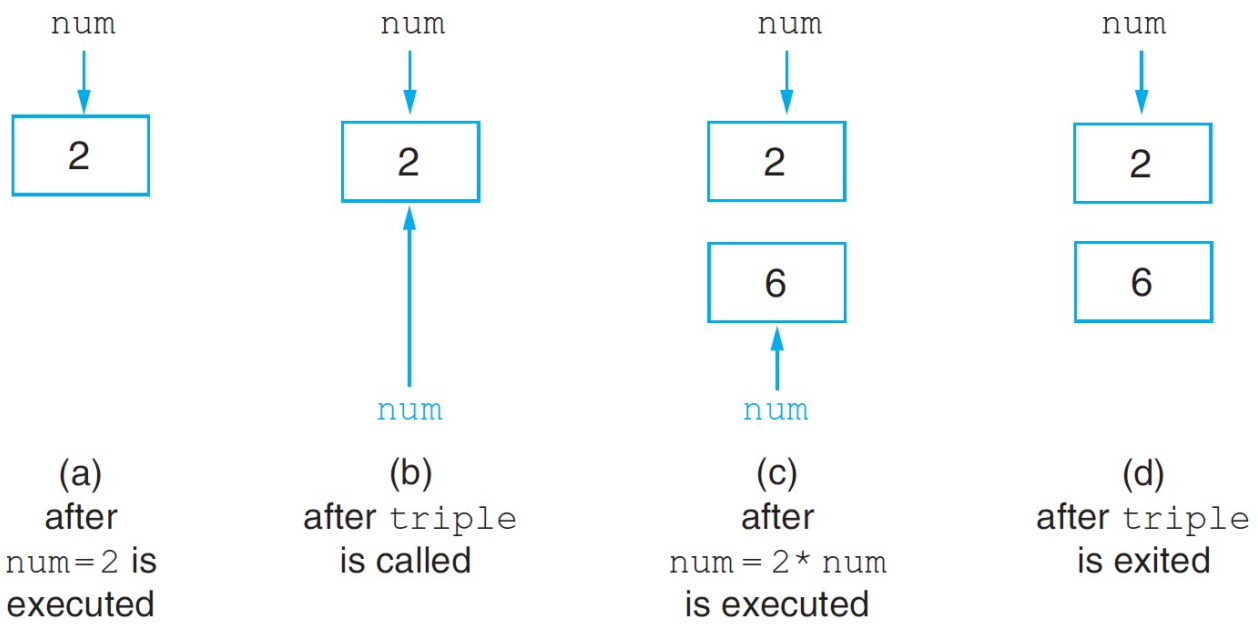

FIGURE 4.2 Passing a value to a function.

Classwork 9: Present Value of Ordinary Annuity

- Define a function that calculates the present value of an ordinary annuity – there is a fixed amount of cash flow at the *end* of each period. Assume the beginning balance is zero.
- The function's arguments will include
  - the fixed amount of cash flow;
  - the discount rate (fixed for each period);
  - the number of periods.
- Use a repetition structure (for loop or while loop) when defining the function.
- The function will be capable of the following. Upload the .py file on Canvas.

```
>>> presentValue(1000, 0.05, 5)
4329.48
```

Present Value of Ordinary Annuity

- *Example*. The following calculates the present value of an annuity over five years with $1000 at the end of each year and a 5% annual discount rate.