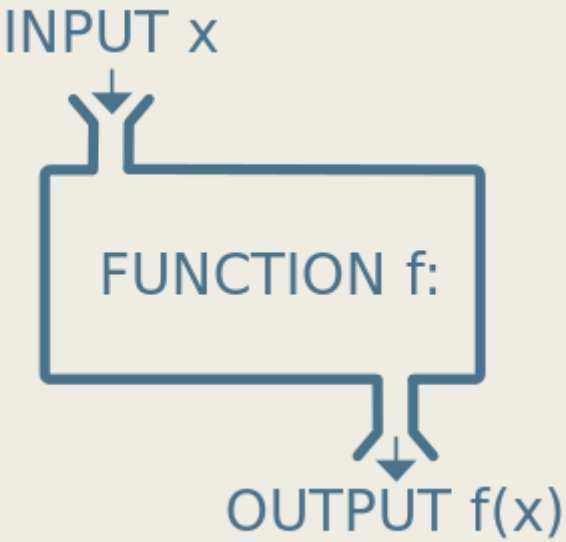
FUNCTIONS

Structured Programming

Why Functions



- **Organization** – A large program is easier to read and modify if it is logically organized into functions.
- **Autonomy** – Programs should be designed so that they consist mainly of stand-alone functions or modules. Each function is autonomous, meaning the function does not depend on data or code outside the function any more than necessary
- **Encapsulation** – It refers to enclosing the details of a function within the function itself, so that those details do not have to be known in order to use the function.
- **Reusability** – Functions should be reused in the same program or even in other programs.

Built-in Function (1 of 2)

- Build-in Functions - https://docs.python.org/3/library/functions.html
  - *Receive input*
  - *Process the input*
  - *Have output*
- 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 Function (2 of 2)

- Output of functions is a single value
  - *Function is said to return its* ***output***
- Items inside parentheses called **arguments**
- https://www.w3schools.com/python/python_ref_functions.asp
- Examples:

```
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
```

Modules

Ex1a_import_modules.py

- A module is a file that contains a collection of related functions
- Before we can use the module, we have to import it:

```
>>> import random
```

- The module object contains the functions and variables defined in the module
- To access one of the functions, you have to specify the name of the module and the name of the function, separated by a dot (also known as a period).
- This format is called dot notation:

```
>>> grade = random.randint(1, 100)
```

Math Module

Ex1b_from_import.py

- Python has built-in math functions and extensive **math module** that perform mathematical tasks on numbers. https://www.w3schools.com/python/module_math.asp
- The standard library module math contains trigonometric, exponential, and logarithmic function, and is regularly used in mathematics, science, and engineering applications.
- The advantage of importing everything from the math module is that your code can be more concise.
- The disadvantage is that there might be conflicts between names defined in different modules, or between a name from a module and one of your variables.

Importing Modules

```
import math
# import math moudule
print(math.pi)
print(math.ceil(math.pi))
print(math.floor(math.pi))
print(round(math.pi))
```

```
from math import pi
print(pi)
```

```
from math import *
# equal to import math
print(pi)
print(cos(pi))
print(sqrt(256))
```

PE9_1 and PE9_2

1. input() & fmod()
a) Use input() function to request any two numbers.
b) Use *math* module, *fmod*() to return the remainder of the user input.
c) Print out the result as an integer.
*d) Implement the validation of denominator to be a non-zero number in a sentinel-controlled loop.

```
Example Output 1

Enter a numerator: 10
Enter a denominator: 2
10 mod 2 = 0
```

```
Example Output 2

Enter a numerator: 1
Enter a denominator: 0
Denominator cannot be zero. Try again.

Enter a numerator: 0
Enter a denominator: 1
0 mod 1 = 0
```

2. randint() & isqrt()
a) Use *random* module, *randint*() to generate a random number in the range (1, 100).
b) Use *math* module, *isqrt*() to round a square root number downwards to the nearest integer.
c) Print out the result.

```
Example Output 1

Square root of 4 = 2
```

```
Example Output 2

Square root of 8 = 2
```

User-defined Functions

- Syntax

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

  - *par1, par2 are variables (called parameters)*
  - *Expression evaluates to a literal of any type*
  - *Header must end with colon*
  - *Each statement in block indented same*
  - *Return statement is optional*

Creating a Function

Ex2_function.py

- Example: No parameter & no return value

```python
# creating a function
def my_function():
    print("Hello")
    print("Python")
    print("World")

# calling a function
my_function()
```

```
Hello
Python
World
```

Calling a Function

- To **call** (or, **invoke**) a function, use the function name followed by parenthesis
- Calling a function will cause the function block to be executed
- Example:

```
…
6       def my_function():
7          print("Hello")
8          print("Python")
9          print("World")
…
22      my_function()
```

User-defined Functions with Parameters

Ex3_function_parameters.py

- Information can be passed to functions as parameter.
- Parameters are specified after the function name, inside the parentheses.
- You can add as many parameters as you want, just separate them with a comma.
- When the function is called, we pass along values of parameters, which are used inside the function.

```
Hello World
Hello Python

Hi
there

3.9
python version
```

```python
def my_function_1(p1):
    print("Hello " + p1)

def my_function_2(p1, p2):
    print()
    print(p1)
    print(p2)

my_function_1("World")
my_function_1('Python')
my_function_2("Hi","there" )
my_function_2(3.9, "python version")
```

PE9_3 and PE9_4

3. Write a function *hello*() that outputs "Hello World" to the console. Implement the code to test the function.

4. Modify the function, *hello*() above with a parameter.
a) Define the function, *helloNo*(n) with a loop to call *hello()* n times to the console.
b) Use the parameter, n for the numbers of iterations in the loop.

```
Example Output: HelloNo(3) will print the following

Hello World
Hello World
Hello World
```

Passing Arguments (1 of 2)

Ex4_pets.py

- Positional Arguments – order matters

```python
def describe_pet(animal_type, pet_name):
    """Display information about a pet."""
    print(f"\nI have a {animal_type}.")
    print(f"My {animal_type}'s name is {pet_name.title()}.")

describe_pet('hamster', 'harry')
describe_pet('dog', 'willie')
describe_pet('harry', 'hamster')
```

```
I have a hamster.
My hamster's name is Harry.

I have a dog.
My dog's name is Willie.

I have a harry.
My harry's name is Hamster.
```

Passing Arguments (2 of 2)

- Keyword Arguments – order does not matter

```python
def describe_pet(animal_type, pet_name):
    """Display information about a pet."""
    print(f"\nI have a {animal_type}.")
    print(f"My {animal_type}'s name is {pet_name.title()}.")

describe_pet('hamster', 'harry')
describe_pet('dog', 'willie')
describe_pet('harry', 'hamster')
describe_pet(pet_name = 'harry', animal_type = 'hamster')
```

```
I have a hamster.
My hamster's name is Harry.

I have a dog.
My dog's name is Willie.

I have a harry.
My harry's name is Hamster.

I have a hamster.
My hamster's name is Harry.
```

Default Parameter Value (1 of 3)

- Parameters of a function can have default values
  - *Assigned to them when no values are passed to them*
- Format for definition using default values

```
def functionName(par1, par2, par3=value3, par4=value4):
```

Default Parameter Value (2 of 3)

Ex5_pets_dog.py

- The default value is used if calling the function without parameter

```python
def describe_pet(pet_name, animal_type='dog'):
    """Display information about a pet."""
    print(f"\nI have a {animal_type}.")
    print(f"My {animal_type}'s name is {pet_name.title()}.")

describe_pet('willie')
```

Python - 2a_pets_dog.py:6 ✓

```
I have a dog.
My dog's name is Willie.
```

Default Parameter Value (3 of 3)

```
def total(w, x, y=10, z=20):
    return (w ** x) + y + z
```

Function Call	Value	Calculated As
`total(2, 3)`	38	$2^3 + 10 + 20$
`total(2, 3, 4)`	32	$2^3 + 4 + 20$
`total(2, 3, 4, 5)`	17	$2^3 + 4 + 5$

Return a Numerical Value

Ex6_return_number.py

- Use the *return* statement to generate a return value of a functions

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

```
def total(w, x, y=10, z=20):
    return (w ** x) + y + z
```

Returning a String Value

Ex7_formatted_name.py

- Making an argument optional

```python
def get_formatted_name(first_name, last_name, middle_name=''):
    """Return a full name, neatly formatted."""
    if middle_name:
        full_name = f"{first_name} {middle_name} {last_name}"
    else:
        full_name = f"{first_name} {last_name}"
    return full_name.title()
```

```python
musician = get_formatted_name('jimi', 'hendrix')
print(musician)

musician = get_formatted_name('john', 'hooker', 'lee')
print(musician)
```

```
Jimi Hendrix
John Lee Hooker
```

Returning a Dictionary

Ex8_person.py

- Making an argument optional

person.py

```python
def build_person(first_name, last_name, age=None):
    """Return a dictionary of information about a person."""
    person = {'first': first_name, 'last': last_name}
    if age:
        person['age'] = age
    return person

musician = build_person('jimi', 'hendrix', age=27)
print(musician)
musician = build_person('Adele', 'Adkins')
print(musician)
```

Python - person.py:12 ✔

```
{'first': 'jimi', 'last': 'hendrix', 'age': 27}
{'first': 'Adele', 'last': 'Adkins'}
```

Returning a String from Input()

Ex9_greeter.py

- Using input() in a while loop

greeter.py

```python
def get_formatted_name(first_name, last_name):
    """Return a full name, neatly formatted."""
    full_name = f"{first_name} {last_name}"
    return full_name.title()

# This is an infinite loop!
while True:
    print("\nPlease tell me your name:")
    print("(enter 'q' at any time to quit)")

    f_name = input("First name: ")
    if f_name == 'q':
        break

    l_name = input("Last name: ")
    if l_name == 'q':
        break

    formatted_name = get_formatted_name(f_name, l_name)
    print(f"\nHello, {formatted_name}!")
```

```
Please tell me your name:
(enter 'q' at any time to quit)
First name: john
Last name: smith

Hello, John Smith!

Please tell me your name:
(enter 'q' at any time to quit)
First name: tom
Last name: cat

Hello, Tom Cat!

Please tell me your name:
(enter 'q' at any time to quit)
First name: jerry
Last name: mouse

Hello, Jerry Mouse!

Please tell me your name:
(enter 'q' at any time to quit)
First name: 123
Last name: q
```

Returning a Boolean Value

Ex10a_boolean_function.py

Functions can return a Boolean value, which is often convenient for hiding complicated tests inside functions.

```
def is_divisible(x, y):
    """function returning a boolean value"""
    return x % y == 0
    '''
    if x % y == 0:
        return True
    else:
        return False
    '''
```

The result of the == operator is a Boolean, so we can write the function more concisely by returning it directly.

```
def is_divisible(x, y):
        return x % y == 0
```

Adding Exception Handling

Ex10b_boolean_function.py

Use exception handling to report and recover from errors that occurs while a program is running

```
#implement the exception handling
try:
    print("Divisible = ", is_divisible(6, 0))
except:
    print("Exception: ZeroDivisionError")
```

PE9_5 and PE9_6

- Write your codes and run

Summary (1 of 3)

- Functions are used to break complex problems into small problem, to eliminate repetitive code, and to make a program easier to read by separating it into logical unit. Also, functions can be reused in other programs.
- Functions name code segments in much the way that the variables name numbers, and lists. Functions allow programmers to focus on the main flow of a complex task and defer the details of implementation. This method of program construction is known as **modular** or **top-down** design. As a rule, a function should perform only one task, or several closely related tasks, and should be kept relatively small.
- The parameters in a function definition are called ***formal parameters*** and then the arguments in a function call are called ***actual parameters***.
- Python has an object called *None* that is used to denote a lack of value, and has no methods.

Summary (2 of 3)

- There are two types of parameters that a function can have – *positional parameters* (also called non-default parameters) and *default parameters*.
- Default parameters have the form **param = defaultValue**. Positional parameters are not followed by an equal sign and a default value. If a function has both parameters, the positional parameter must precede the default parameters.
- There are two types of arguments that can appear in a function call – *positional arguments* (also *called non-keyword arguments*) and *keyword arguments*. Keyword arguments have the form **parameterName = value**, where *value* is an expression. Positional arguments consist solely of an expression. If a function call has both types of arguments, the positional arguments must precede the keyword arguments.

Summary (3 of 3)

- The order of positional arguments is most important; order is not important for keyword arguments.
- The first function will be named *main* and sometimes will be preceded by **import** statements and global variables. All program will end with the statement **main()** to call the program's *main* function.
- The function *main* should not perform lengthy computations. Ideally *main* should be a supervisory function calling other functions according to the application's logic.

Quiz 9

- Quiz 9A has 10 questions in 15 minutes, 10 pts
  - *10 multiple choice/true or false questions, 1 pt. for each question*
  - *Quiz 9A has two attempt, the higher grade will be selected*
  - *Submit Quiz 9A (at least 1-minute)* ***before*** *the due time to Blackboard*

- Quiz 9B has 2 code questions, 15 pts
  - *Write the Python code based on the given question*
  - *Each question will be given during the first 10-minute of each session of week 9*
  - *Quiz 9B-1 on session A, and Quiz 9B-2 on session B*
  - *Quiz 9B has one attempt*

DB 9

- Instruction:
  1) **Modify** any **three** questions from **PE9_1 to PE9_5** to whatever you want for practice. Make sure to **comment** on your **modification** (1.2 pt).
  2) Simplify the following function (0.3 pt).

```
def f(x):
    if x >0:
        return True
    else:
        return False
```

  3) Submit your posts before the due date. Let's learn from each other.