SFU SIMON FRASER UNIVERSITY
ENGAGING THE WORLD

CMPT 120: Introduction to Computing Science and Programming 1

Functions

python™

Reminders

One-Stop Access To Course Information

- **Course website: One-stop access** to all course information.

  http://www2.cs.sfu.ca/CourseCentral/120/liaqata/WebSite/index.html

- Course Outline
- Exam Schedule
- Python Info
- CourSys/Canvas link
- Learning Outcomes
- Office Hours
- Textbook links
- and more…
- Grading Scheme
- Lab/Tutorial Info
- Assignments

- **Canvas**: Discussions forum - https://canvas.sfu.ca/courses/39187
- **CourSys**: Assignments submission, grades - www.coursys.sfu.ca

Course Topics

1. General introduction
2. Algorithms, flow charts and pseudocode
3. Procedural programming in Python
4. Data types and Control Structures
5. Binary encodings
6. Fundamental algorithms
7. Basics of (Functions and) Recursion (Turtle Graphics)
8. Basics of computability and complexity
9. Subject to time availability:
   - Basics of Data File management

Today's Topics

1. Function: In-Class Code
   - Defining and Calling a Void Function
   - Defining and Calling a Value-Returning Function
2. Generating Random Numbers
3. Using the math Module
4. Storing Functions in Modules
5. Turtle Graphics: Module Approach

Defining and Calling a Void Function

- Write a Python program **calc.py** that
  1. Defines and calls a menu function.
  2. The function prints the following lines and **do not** return any value:
     Enter A to add numbers:
     Enter S to subtract numbers:

```
# calc.py
# define a menu function
def menu():
    print("Enter A to add numbers: ")
    print("Enter S to subtract numbers: ")
# call the menu function
menu()
```

- Write a Python program **circ.py** that
  1. Draws a circle for given diameter(25).
  2. The function **do not** return any value.
  3. Call the function to circle of diameter 50.

```
# circ.py
# define a circle function
def circle(diameter):
    turtle.circle(diameter)
# call the circle function
import turtle
circle(25)
circle(50)
```

Defining and Calling a Value-Returning Function

- Write a Python program **calc.py** that
  1. Defines and calls an **add** function.
  2. The function adds two given numbers and **returns** the sum value.

```
# calc.py
# define the add function
def add( num1, num2):
    sum = num1 + num2
    return sum
# call the add function
result = add(56, 78)
```

- Write a Python program **enroll.py** that
  1. defines and calls a **name** function.
  2. The function inputs first name and last name. It returns both first and last names.

```
# enroll.py
# define the name function
def name():
    fname = input("Enter first name: ")
    sname = input("Enter second name: ")
    return fname, sname
# call the name function
f_name, s_name = name()
```

Generating Random Numbers

- Random number are useful in a lot of programming tasks
- Python includes a module called **random** for working with random numbers.
- The **random** module includes various functions to generate random numbers.
  - randint() randrange() random() uniform()
- Import the **random** module to use (call) the random functions.
  - Use of module requires an `import random` statement.
  - Format: **module_name.function_name()**

random.randint()
random.randint(1, 10)
number = random.randint(1, 10)

Random Number Functions

- **randint()**: generates a random number in the range provided by the arguments.
- **randrange**: similar to range function, but returns randomly selected integer from the specified range:

random.randrange(10)	**For example: 9**
random.randrange(11, 30)	**For example: 25**
random.randrange(100, 200, 5)	**For example: 155**

- **random** function: It returns a random float in the range of 0.0 and 1.0
  - The random function does not receive any arguments.
- **uniform** function: returns a random float but allows user to specify range.

Random Number Seeds

- Random number functions use clock time as a seed value.
- We can specify our own seed value.
- random.seed()
- random.seed(10)
- A seed value initializes the function.
- Same seed value generate a same set of random numbers.

The `math` Module

- **`math` module**: A part of standard library that contains functions for performing mathematical calculations.
  - Typically accept one or more values as arguments, perform mathematical operation, and return the result
  - Use of module requires an `import math` statement.
  - Example:

```
circle_area = math.pi * radius**2
```

The `math` Module

`math` Function	Description
`acos(x)`	Returns the arc cosine of `x`, in radians.
`asin(x)`	Returns the arc sine of `x`, in radians.
`atan(x)`	Returns the arc tangent of `x`, in radians.
`ceil(x)`	Returns the smallest integer that is greater than or equal to `x`.
`cos(x)`	Returns the cosine of `x` in radians.
`degrees(x)`	Assuming `x` is an angle in radians, the function returns the angle converted to degrees.
`exp(x)`	Returns e^x
`floor(x)`	Returns the largest integer that is less than or equal to `x`.
`hypot(x, y)`	Returns the length of a hypotenuse that extends from (0, 0) to (`x`, `y`).
`log(x)`	Returns the natural logarithm of `x`.
`log10(x)`	Returns the base-10 logarithm of `x`.
`radians(x)`	Assuming `x` is an angle in degrees, the function returns the angle converted to radians.
`sin(x)`	Returns the sine of `x` in radians.
`sqrt(x)`	Returns the square root of `x`.
`tan(x)`	Returns the tangent of `x` in radians.

Storing Functions in Modules

- **Modularization**: Grouping of related functions in modules (Python files) for better organization.
  - Makes program easier to understand, test, and maintain.
  - Make it easier to reuse code for multiple different programs.
  - We import the required modules in the program.
- Module is a file that contains Python code.
  - Contains function **definition** but **does not contain calls** to the functions.
    - **Importing programs will call the functions.**
- Rules for module names:
  - File name should end in `.py`
  - Cannot be the same as a Python keyword
- Import module using `import` statement

Storing Functions in Modules: Example

```
# circle.py
#The circle module has functions that perform
# calculations related to circles.
import math
# The area function accepts a circle's radius as an
# argument and returns the area of the circle.
def area(radius):
    return math.pi * radius**2
# The circumference function accepts a circle's
# radius and returns the circle's circumference.
def circumference(radius):
    return 2 * math.pi * radius
```

```
# rectangle.py
# The rectangle module has functions that perform
# calculations related to rectangles.
# The area function accepts a rectangle's width and
# length as arguments and returns the rectangle's area.
def area(width, length):
    return width * length
# The perimeter function accepts a rectangle's width
# and length as arguments and returns the
# rectangle's perimeter.
def perimeter(width, length):
    return 2 * (width + length)
```

Storing Functions in Modules: Example

```
import circle
Import rectangle

circ_area = circle.area(10)
circ_circum = circle.circumference(10)

rect_area = rectangle.area(10)
rect_peri = rectangle. perimeter(10)
```

Liaqat Ali, 2018: Adapted from:

Menu Driven Programs

- **Menu-driven program**: displays a list of operations on the screen, allowing user to select the desired operation
  - List of operations displayed on the screen is called a *menu*
- Program uses a decision structure to determine the selected menu option and required operation.
  - Typically repeats until the user quits.
  - See: **geometry.py** program.

Turtle Graphics: Modularizing Code with Functions

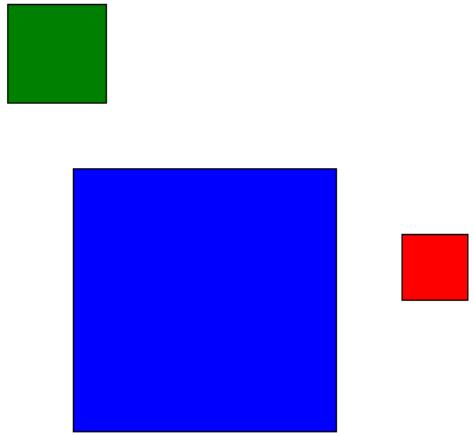
- Commonly needed turtle graphics operations can be stored in functions and then called whenever needed.
- For example, the following function draws a square. The parameters specify the location, width, and color.

```
def square(x, y, width, color):
    turtle.penup()              # Raise the pen
    turtle.goto(x, y)           # Move to (X,Y)
    turtle.fillcolor(color)     # Set the fill color
    turtle.pendown()            # Lower the pen
    turtle.begin_fill()         # Start filling
    for count in range(4):      # Draw a square
        turtle.forward(width)
        turtle.left(90)
    turtle.end_fill()           # End filling
```


Turtle Graphics: Modularizing Code with Functions

- The following code calls the previously shown `square` function to draw three squares:

```
square(100, 0, 50, 'red')
square(-150, -100, 200, 'blue')
square(-200, 150, 75, 'green')
```



Turtle Graphics: Modularizing Code with Functions

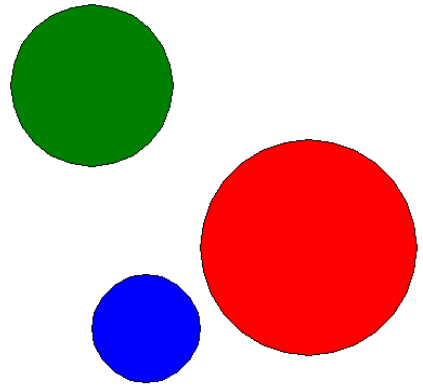
- The following function draws a circle. The parameters specify the location, radius, and color.

```
def circle(x, y, radius, color):
    turtle.penup()                # Raise the pen
    turtle.goto(x, y - radius)    # Position the turtle
    turtle.fillcolor(color)       # Set the fill color
    turtle.pendown()              # Lower the pen
    turtle.begin_fill()           # Start filling
    turtle.circle(radius)         # Draw a circle
    turtle.end_fill()             # End filling
```


Turtle Graphics: Modularizing Code with Functions

- The following code calls the previously shown `circle` function to draw three circles:

```
circle(0, 0, 100, 'red')
circle(-150, -75, 50, 'blue')
circle(-200, 150, 75, 'green')
```



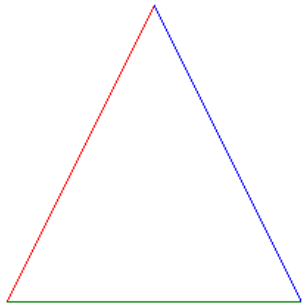
Turtle Graphics: Modularizing Code with Functions

- The following function draws a line. The parameters specify the starting and ending locations, and color.

```
def line(startX, startY, endX, endY, color):
    turtle.penup()                # Raise the pen
    turtle.goto(startX, startY)   # Move to the starting point
    turtle.pendown()              # Lower the pen
    turtle.pencolor(color)        # Set the pen color
    turtle.goto(endX, endY)       # Draw a square
```


Turtle Graphics: Modularizing Code with Functions

- The following code calls the previously shown `line` function to draw a triangle:



```
TOP_X = 0
TOP_Y = 100
BASE_LEFT_X = -100
BASE_LEFT_Y = -100
BASE_RIGHT_X = 100
BASE_RIGHT_Y = -100
line(TOP_X, TOP_Y, BASE_LEFT_X, BASE_LEFT_Y, 'red')
line(TOP_X, TOP_Y, BASE_RIGHT_X, BASE_RIGHT_Y, 'blue')
line(BASE_LEFT_X, BASE_LEFT_Y, BASE_RIGHT_X, BASE_RIGHT_Y, 'green')
```


Questions?