IST 1025

Introduction to Programming

Defining Functions

What Is a Function?

A function is a chunk of code that can be called by name wherever we want to run that code

```
def sqr(n):                 # Definition
    return n ** 2


…

print(sqr(2))               # Call: Displays 4

print(sqr(33))              # Call: Displays 1089

print(sqr(etc))             # Call: Displays whatever
```

Using Functions: Combination

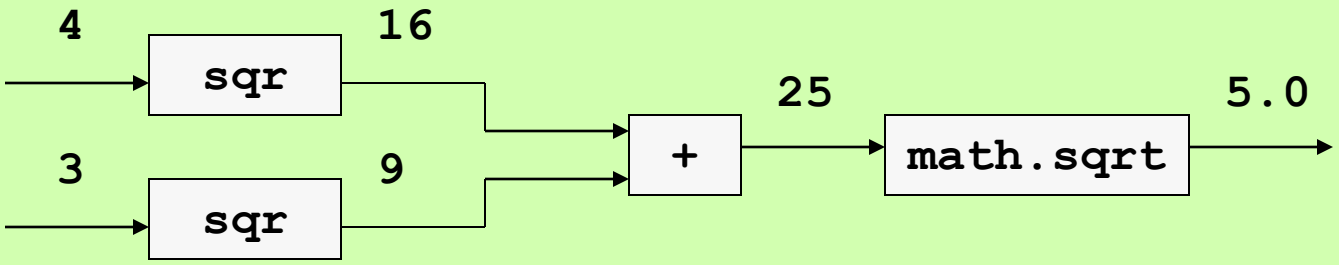
Functions can be used to compute values, wherever operand expressions are expected

```
a = sqr(4)
b = sqr(3)
c = math.sqrt(a + b)
```

Using Functions: Combination

Functions can be used to compute values, wherever operand expressions are expected

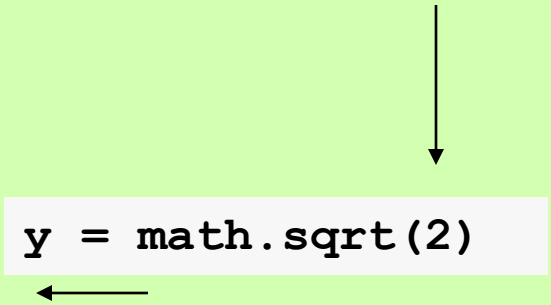
```
a = sqr(4)
b = sqr(3)
c = math.sqrt(a + b)

# Or use function calls as operands:

c = math.sqrt(sqr(4) + sqr(3))
```



Arguments and Return Values

- A function can receive data from its caller (arguments)

- A function can return a single value to its caller



Programmer-Defined Functions

- A function allows the programmer to define a general algorithm in one place and use it in many other places (avoid repetitive patterns)

- A function replaces many lines of code with a single name (abstraction principle)

Function Definition Syntax: Parameters and **`return`** Statements

The *function header* includes 0 or more *parameter* names

```
def sqr(n):                 # Definition
    return n * n
```

```
def <function name>(<param name>, …, <param name>):
    <sequence of statements>
```

The **`return`** statement exits the function call with a value

`return` Statements

If you do not include a **`return`** statement, your function returns the value **`None`**

```
def sqr(n):                    # Definition
    n * n
```

```
>>> print(sqr(33))
None
```

A General Input Function

Define a function that obtains a valid input integer from the user

The function expects a string prompt and the lower and upper bounds of the range of valid integers as arguments

The function continues to take inputs until a valid number is entered; if an invalid integer is entered, the function prints an error message

The function returns the valid integer

Example Use

Pretend that the function has already been defined and imagine its intended use

```
>>> rate = getValidInteger("Enter the rate: ", 1, 100)
Enter the rate: 120
Error: the number must range from 1 through 100
Enter the rate: 99
>>> rate
99
```

```
>>> size = getValidInteger("Enter the size: ", 1, 10)
Enter the size: 15
Error: the number must range from 1 through 10
Enter the size: 5
>>> size
5
```

Definition

```
def getValidInteger(prompt, lower, upper):
    """Repeatedly inputs an integer until that
    integer is within the given range."""
```

A function definition should include a ***docstring***

`help(getValidInteger)` displays this information

Definition

```
def getValidInteger(prompt, lower, upper):
    """Repeatedly inputs a integer until that
    integer is within the given range."""
    while True:
        number = int(input(prompt))
        if number < lower or number > upper:
            print("Error: the number must range from " + \
                  str(lower) + " through " + str(upper))
        else:
            return number
```

The **`return`** statement exits both the loop and the function call

The **`\`** symbol is used to break a line of Python code

Good Programming Practice

- Try to limit the names used in a function to its parameters (data) and other function calls
- Each function should perform a single, coherent task (described in its docstring)
- Try to aim for general methods, using parameters for special cases

Data Encryption Revisited

```
>>> print(encrypt("Exam Friday!"))
69 120 97 109 32 70 114 105 100 97 121 33
```

```
def encrypt(source):
    """Builds and returns an encrypted version of
    the source string."""
    code = ""
    for ch in source:
        code = code + str(ord(ch)) + " "
    return code
```

`source` is a *parameter* and **`code`** and **`ch`** are *temporary variables*

They are visible only within the body of the function

Data Decryption Revisited

```
>>> print(decrypt(encrypt("Exam Friday!")))
Exam Friday!
```

```
def decrypt(code):
    """Builds and returns a decrypted version of
    the code string."""
    source = ""
    for word in code.split():
        source = source + chr(int(word))
    return source
```

Organize Code with a `main` Function

```
import math

def main():
    radius = float(input('Enter the radius: '))
    area = math.pi * radius ** 2
    print('The area is', area, 'square units')

main()  # run this function when this module is imported
        # or launched as a script
```

Example: Is It a Script?

- Run a Python module as a script
- Import it as a module but don't execute the **`main`** function
- Need to ask a question and then take action depending on the answer

The `circlearea` Script

```
import math

def main():
    radius = float(input('Enter the radius: '))
    area = math.pi * radius ** 2
    print('The area is', area, 'square units')

main()  # run this function when this module is imported
        # or launched as a script
```

The **`circlearea`** Script

```
import math

def main():
    radius = float(input('Enter the radius: '))
    area = math.pi * radius ** 2
    print('The area is', area, 'square units')

if __name__ == '__main__':
    main()
```

Each module includes a built-in **`__name__`** variable

This variable is automatically set to **`'__main__'`** if the module is run as a script

Otherwise, this variable is set to the module's name