TCSS 142 – Introduction to Programming

Autumn 2014
Day 11

Day 11 Overview

- Modularization

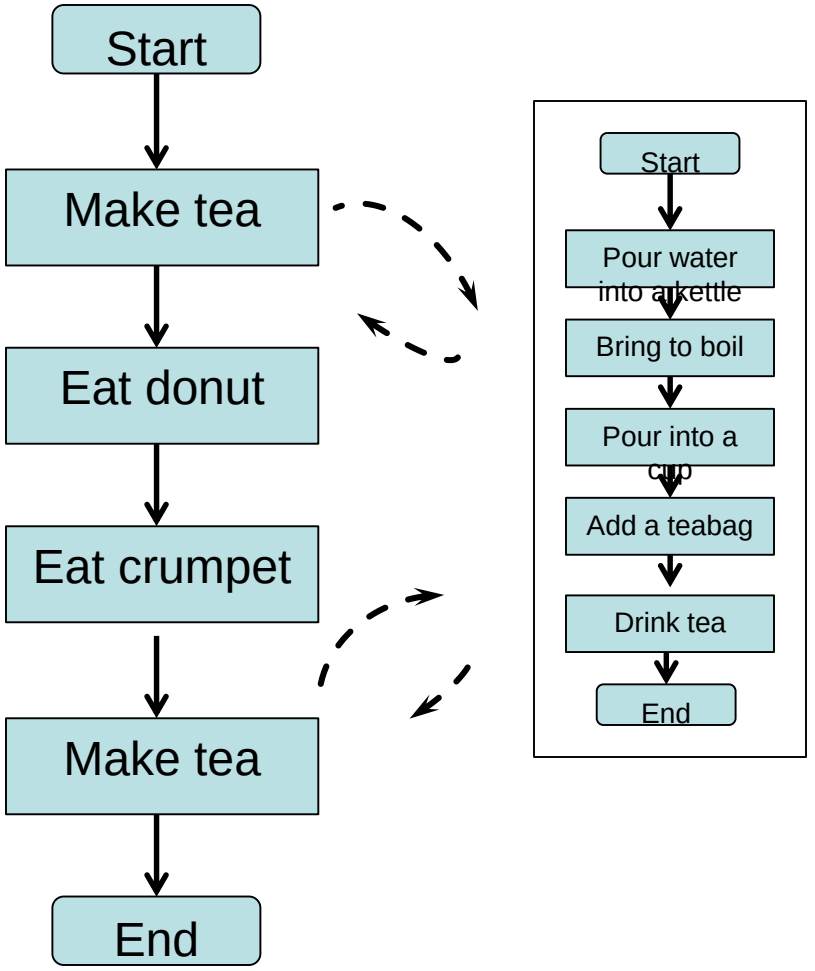
- Function definitions and calls
  - Void
  - Local variables
  - Parameter passing

Functions

- Function: a group of statements within a program that performs a specific task
- Function definition: specifies what a function does

```
def function_name():
    statement
    statement
```

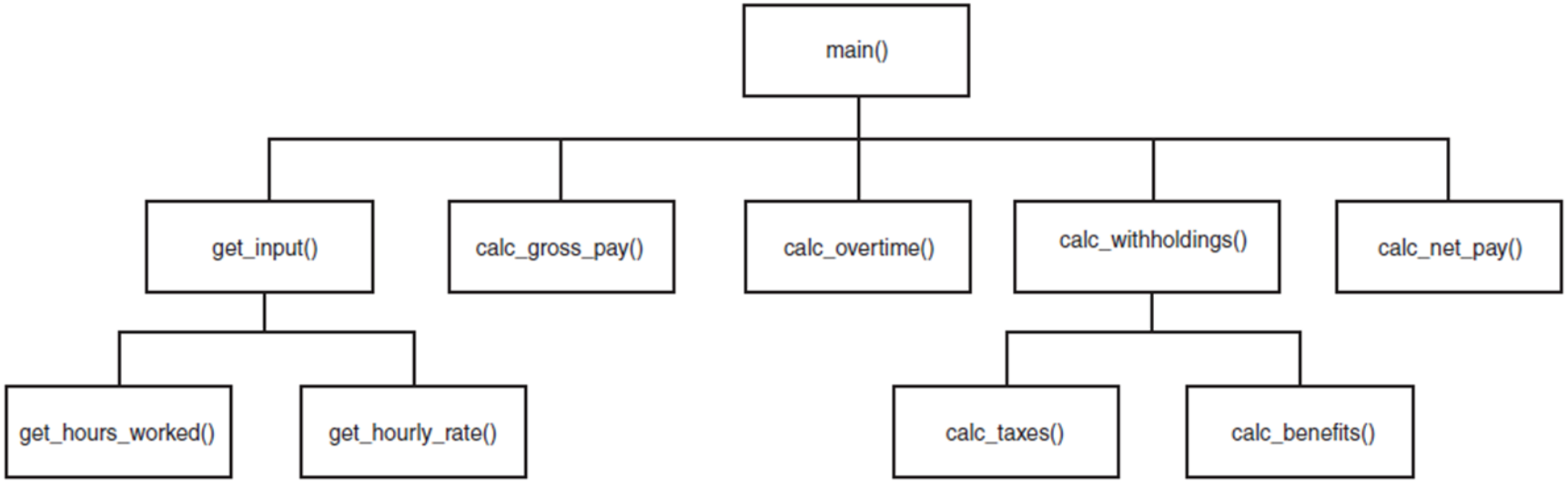
- A function name has to follow identifier naming rules (same as with variables)



Modularization

- Modularized program: program wherein each task within the program is in its own function

Figure 5-10 A hierarchy chart



Procedural Decomposition

- **top-down design:** technique for breaking algorithm into functions
- **procedural decomposition**: dividing a problem into tasks
- **stepwise refinement**: the process of producing a program in stages, adding new functionality at each step

Using Functions

1. Design the algorithm.
   - Look at the structure, and which commands are repeated.
   - Decide what are the important overall tasks.

2. **Declare** (write down) the functions.
   - Arrange statements into groups and give each group a name.

3. **Call** (run) the functions.
   - So far, we have been writing the code line by line, now our main part of the program will be in a function called main, which will execute other functions, if any

Design of an algorithm

```
# Step 1: Make the cake batter.
print("Mix the dry ingredients.")
print("Cream the butter and sugar.")
print("Beat in the eggs.")
print("Stir in the dry ingredients.")
print()
# Step 2a: Bake cookies (first batch).
print("Set the oven temperature.")
print("Set the timer.")
print("Place a batch of cookies into the oven.")
print("Allow the cookies to bake.")
print()
# Step 2b: Bake cookies (second batch).
print("Set the oven temperature.")
print("Set the timer.")
print("Place a batch of cookies into the oven.")
print("Allow the cookies to bake.")
print()
# Step 3: Decorate the cookies.
print("Mix ingredients for frosting.")
print("Spread frosting and sprinkles.")
```

Final cookie program

```
# Step 1: Make the cake batter.
def makeBatter():
    print("Mix the dry ingredients.")
    print("Cream the butter and sugar.")
    print("Beat in the eggs.")
    print("Stir in the dry ingredients.")
    print()

# Step 2: Bake cookies.
def bakeCookies():
    print("Set the oven temperature.")
    print("Set the timer.")
    print("Place a batch of cookies into the oven.")
    print("Allow the cookies to bake.")
    print()
```

Final cookie program

```
# Step 3: Decorate the cookies.
def decorateCookies():
    print("Mix ingredients for frosting.")
    print("Spread frosting and sprinkles.")
    print()

def main():
    makeBatter()
    bakeCookies()
    bakeCookies()
    decorateCookies()

main()
```

Calling a Function

Executes the function's code

- Syntax:

```
name()
```

  - You can call the same function many times if you like
  - One function can call other functions
  - It can even call itself (recursion)

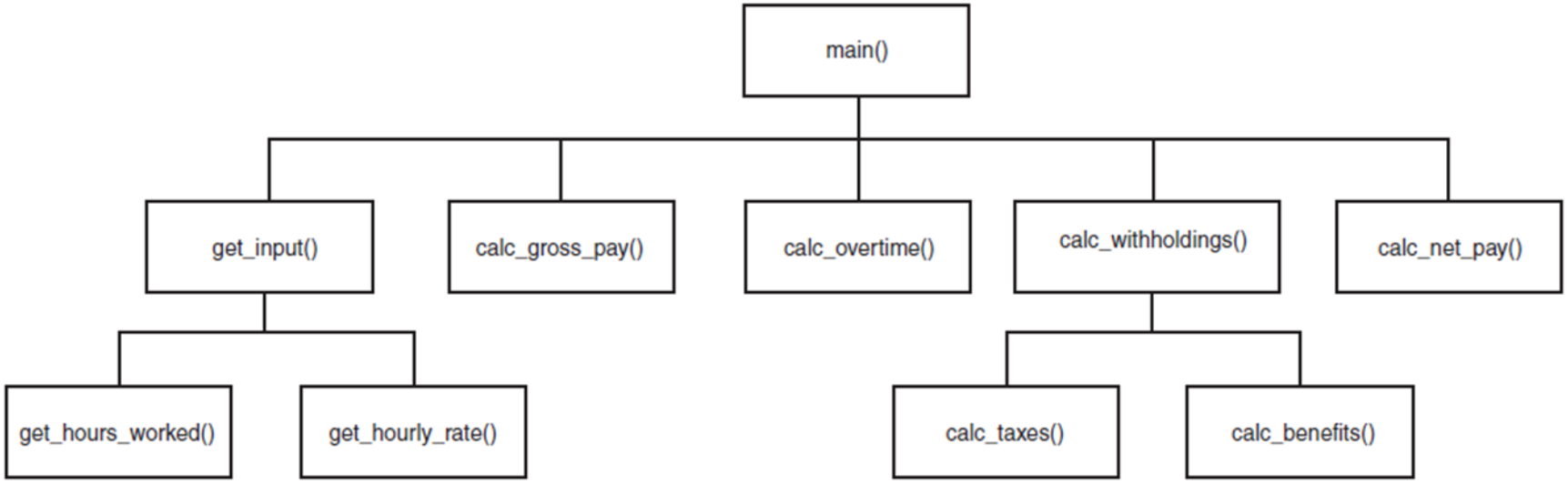
When to Use Functions

- Place statements into a function if:
  - The statements are related structurally, and/or
  - The statements are repeated.

- You should not create functions for:
  - An individual print statement.
  - Only blank lines. (Put blank print in main.)
  - Unrelated or weakly related statements. (Consider splitting them into two smaller functions.)

Functions Calling Functions

Figure 5-10 A hierarchy chart



Functions Calling Functions

```
def main():
    message1()
    message2()
    print("Done with main.")

def message1():
    print("This is message1.")

def message2():
    print("This is message2.")
    message1()
    print("Done with message2.")

main()
```

Control flow

- When a function is called, the program's execution...
  - "jumps" into that function, executing its statements, then
  - "jumps" back to the point where the function was called.

```
def main():
    message1()

    message2()



    print("Done with main.")



    ...
```

```
message1():
    print("This is message1.")
}
```

```
message2():
    print("This is message2.")
    message1()

    print("Done with message2.")
```

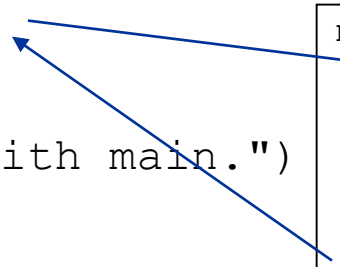
Control flow

- When a method is called, the program's execution...
  - "jumps" into that function, executing its statements, then
  - "jumps" back to the point where the function was called.

```
def main():
    message1()

    message2()



    print("Done with main.")



    ...
```

```
message1():
    print("This is message1.")
}
```

```
message2():
    print("This is message2.")
    message1()

    print("Done with message2.")
```



Exercise

- Download the program `fightSong.py`. Run it to see the output produced by the program. The program has poor structure and redundancy. Crete a copy of the program called `fightSongBetter.py` and restructure it by adding main and at least two other functions.

- Can you do better? main + 3 functions

Local Variables

- Local variable: variable that is assigned a value inside a function
  - Belongs to the function in which it was created
  - Only statements inside that function can access it, error will occur if another function tries to access the variable
- Local variable cannot be accessed by statements inside its function which precede its creation.
- Different functions may have local variables with the same name
  - Each function does not see the other function's local variables, so no confusion

Scope Implications

- **scope**: The part of a program where a variable is visible

```
def main():
    size = 4        # size variable local to main
    func1()
    func2()

def func1():
    for var in range(size):
     # ERROR: size not visible
        ...
```

Scope Implications

```
def func2():
    size = 18  # size variable local to func2
    for z in range(size):
      # ok, refers to local size
        print(message)
      # ERROR: message not created yet

    message = "I will crash your program"
    # message variable local to func2
```

- Problem: so what do we do if we want a variable visible in a function?

Parameter Passing

- Argument vs parameter

Figure 5-14 The `value` variable and the `number` parameter reference the same value

```
def main():
    value = 5
    show_double(value)


def show_double(number):
    result = number * 2
    print(result)
```

- Some arguments can be modified inside the function, some cannot – depends on the type

Parameter Passing

Figure 5-14 The `value` variable and the `number` parameter reference the same value

```
def main():
    value = 5
    show_double(value)


def show_double(number):
    result = number * 2
    print(result)
```

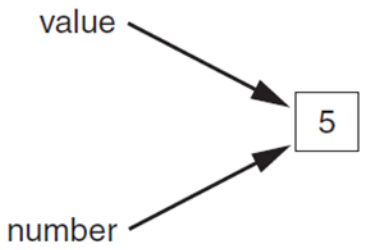
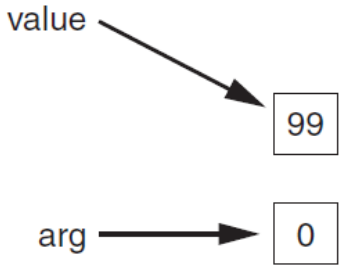


Figure 5-18 The `value` variable is passed to the `change_me` function

```
def main():
    value = 99
    print('The value is', value)
    change_me(value)
    print('Back in main the value is', value)


def change_me(arg):
    print('I am changing the value.')
    arg = 0
    print('Now the value is', arg)
```



Example

```
def strange(x):
    x = x + 1
    print("1. x = ", x)

def main():
    x = 25
    strange(x)
    print("2. x = ", x)

main()
```

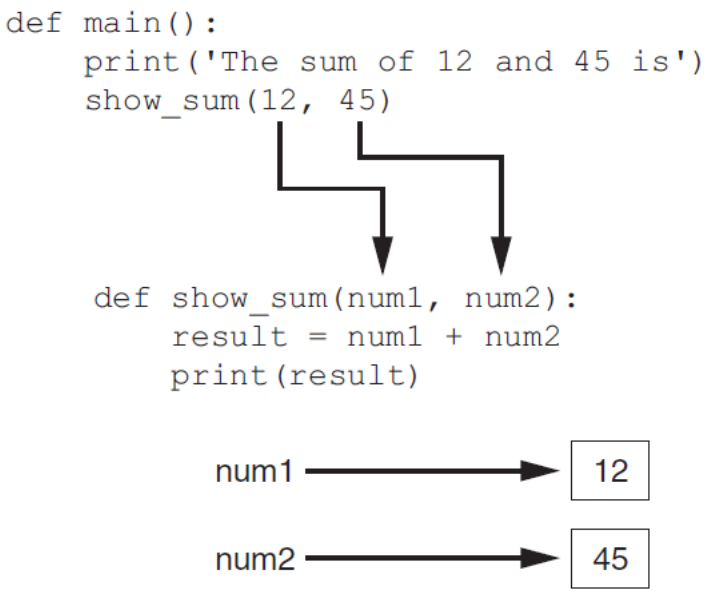
Output:

??

Multiple Parameters

- Arguments are passed by position to corresponding parameters

Figure 5-16 Two arguments passed to two parameters



Example

- Let's rewrite our `whileExample.py` program so that radius and choice are read in in one function and then passed to another function. Rename the program `params.py`

- Steps:
  - Put an entire program into a function called main and then call main
  - Create a function called `calculate(radius, choice)` and move appropriate code into its definition
  - Call calculate from main

Parameter Mystery

```
def mystery(x, z, y):
    print(z, "and", y - x)

def main():
    x = 9
    y = 2
    z = 5

    # call mystery with z, y, x
    # call mystery with y, x, z

main()
```

Output:

??

Parameters and Loops

- A parameter can guide the number of repetitions of a loop.

```
import random
def chant(n):
    for i in range(n):
        print("This is the song that never ends…")
        print("Yes, it goes on and on my friends.")

def main():
        var = 10
        chant(1000)
        chant(var)
        chant(random.randint(1,100))
        chant(var * var)
```

Keyword Arguments

- In Python, there exist keyword arguments – arguments that specify which parameter the value should be passed to
  - Position irrelevant
  - General format

- Download `roses.py` from `Canvas`
  - Let's examine, comment, and run

Last Slide ☺

- Read the rest of chapter 5 and complete the quiz by the next class meeting on Tuesday.

- Class ends at 17:10