**Reminder: students should not see this document, don’t provide it electronically to them but also it’s not a good idea to show it during a screen share because you may end up showing something that they shouldn’t see.**

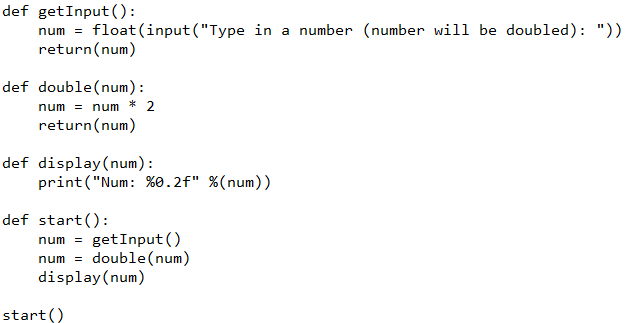
Spring/summer, Third week: second tutorial (Wednesday or Thursday), first hour

Fall/winter: Sixth week, first tutorial

* Go over Mini-A2b:
  + Similar to the previous program knowing your terminology is essential before starting this assignment .Write a program that consists of 4 functions:

|  |  |  |  |
| --- | --- | --- | --- |
| **Function name** | **Inputs/parameters** | **Return values** | **Responsibility of the function** |
| getInput() | None | Float (numeric assumed) | Prompt for and gets the user to enter a number, return that number back to the caller of the function. |
| double() | Float (numeric assumed) | Float (numeric assumed) | Take a number as a parameter, return the double of that number. |
| display() | Float (numeric assumed) | None | Take a number as a parameter, display the number to the console (text-output) with 2 places of precision. |
| start() | None | None | Starting execution point of the program, contains the calls to the other 3 functions |

* + **Solution (obviously you shouldn’t show this in class I’ve included it just so you could visualize the entire program and what was required)**



* + **Terminology to go over**
    - **Function call:** Running the instructions in a function (the following program runs print twice and input once, 3 function calls in all below). They should be quite familiar with this with some of python’s pre-created functions so this is a reminder of they should already know.

print()

input("Enter a number: ")

print("Hello" + " there")

* + - **Function arguments/parameters/inputs** (for this course they will be 3 terms that describe the same thing – we won’t distinguish between formal parameters and actual arguments this semester). You can use this remember of what they already know when you teach them new concepts.

Print() # No argument

Input("Enter a number: ") # 1 string argumement

Print("Hello" + " there") # 1 string argument (2 strings

concatenated)

* + - * Input(string) : takes 1 string argument
      * Print(<string>) : no parameters passed in still results in an automatic \n
      * Other functions may take more or less than 1 argument
      * Pow(base,exponent) : 2
      * Round(number,numDigits) : 1 or 2 arguments
      * Note: inputs to a function does not refer to user input! Make a clear distinction because students sometimes confuse these cases.

Def fun(num):

Vs.

Def fun():

Num = int(input(“Type in a number: “))

* + - * E.g. the function pow takes as input 2 number does not mean it works by asking the user to type into the keyboard 2 numbers when the function runs
    - **Function return values**: a function returns either 0 or 1 values (no more than 1 although it may appear to be return more than one we’ll show them later how only one item – a tuple – is what’s returned in python…)
      * Print: no return value to store
      * Input: returns a value but the programmer must store the value return. Contrast

num1 = -1

num2 = -1

num1 = input("Enter a number: ")

input("Enter a number: ")

print(num1,num2)

* + Mini-A2a and mini-A2b have extremely strict requirments but remind students they must know the terminology used in order to understand what they must do.

Stress to students: That means they must actually study (and ideally try to apply) the concepts taught I the functional decomposition lectures and tutorials **before** trying work on the assignment (not all students do this and try to learn it at the same time).

* Go over two versions of a hello world program (in one version the function call to the first/start function is omitted)
  + Learning lesson: no function in Python even the starting function is automatically executed.
  + Instead each function must be explicitly called (this will trip up experienced programmers more than the novices)
  + **Program 1**: 1fun\_sayHello.py

def sayHello():

print(""""

Hello! Hi! Sup? Wei? Na ho? Guten taag! Kamoustaka! Salaam! Shaloom! Buenos! Bonjour!""")

sayHello()

* + **Program 2**: 2fun\_hello.py
    - (Run the program and have them try to figure out why nothing appeared onscreen)

def sayHello():

print(""""

Hello! Hi! Sup? Wei? Na ho? Guten taag! Kamoustaka! Salaam! Shaloom! Buenos! Bonjour!""")

(The function call for sayHello() was omitted)

* + **Students do: First exercise** (It’s so trivial that no there is no online solution):
    - Write a function that will display their name onscreen, get them to do it in steps: let them try step 1, show the solution, let them try step 2, show the solution.

**# Step 1, define function**

def sayName():

print(“James Tam”) # Their name not my name

**# Step 2, call function**

sayName()

* + Declaring local variables in a function (a local to the function, not passed as a parameter)
    - **Program 3**: 3fun\_locals.py

def fun():

print(num) # Not OK 'num' out of scope

def start():

num = 12

print (num) # OK within scope

fun()

* + **Students do: second exercise**: exercise2\_defining\_functions\_globals\_locals.py
    - Declare a global constant called HUMAN\_CAT\_AGE\_RATIO that is set to 7.
    - Write a function called ‘catAge’ that will prompt for the age in years.
    - The age should be stored in local variable.
    - The function then calculates the cat age (human age multipled by 7 – but the named constant rather than an unnamed constant should be used).
    - Write a call to the function that you just defined.
    - **Solution (formatting of the output need not be identical, just clear and reasonable neat)**

HUMAN\_CAT\_AGE\_RATIO = 7

def catAge():

humanAge = float(input("Type in your age in years: "))

catAge = humanAge \* HUMAN\_CAT\_AGE\_RATIO

print("Human years: %d,\tCat years: %d"

%(humanAge,catAge))

catAge()

* Simple parameter passing (alternate terms can be used: function parameters, arguments, inputs – but again distinguish between inputs to a function and getting the user to type in input).
  + **Program 4, parameter passing**: 4fun\_parameters.py

def fun(num): # Copy the value of 12 into a local variable

print(num)

def start():

num = 12

print (num) #OK num is within scope

fun(num)

* **Students do: Second exercise** (‘exercise3\_starting.py’)
  + Modify the starting program

def start():

name = input("Enter your name: ")

* + Define a function ‘displayName()’ that takes the ‘name’ variable as an input/parameter and displays it onscreen. Include the function call in your solution.
  + Give them longer with this one.
  + Give them a few minutes and then give a piece of the solution. Give them a few more minutes to continue working after they get a piece.
  + **Step 1**: Define the function (don’t let someone blurt out the obvious error (caution if you allow public chat or allow people to talk out of turn) – at least it may be obvious to some but others should work it through on their own. Instead let the class try typing in this version and give everyone a chance to interpret the error message before someone spews out an answer)

def displayName():

print(name)

* + **Step 2**: Pass the argument into the function

def displayName(name):

print(name)

* + **Step 3**: Call the function

def start():

name = input("Enter your name: ")

displayName(name)

* **Full solution.**

def displayName(name)

print(name)

def start():

name = input("Enter your name: ")

displayName(name)

start()

* Function return values
  + **Program 5, function return values**: 5fun\_return.py

def fun():

num = int(input("Enter a number: ")) #num is local to fun

return(num) #return allows access to the data stored in num

def start():

num = fun() #Copy the value returned from fun into a num that's local to start

print(num)

start()

Spring/summer, Third week: second tutorial (Wednesday or Thursday), second hour

Fall/winter: Sixth week, second tutorial

* Pass by value: The parameter passing mechanism that applies for many types in Python (local copy made of parameter)
  + **First example**: 1fun\_parameter\_pass\_copy.py

def fun(num): # Copy the value of 12 into a local variable

print(num)

num = num + num # Only local copy changed

print(num)

def start():

num = 12

print(num) # OK within scope

fun(num)

print(num) # Variable local to 'start' unchanged

start()

* How to access locals after a function has ended: ‘Returning’ the contents of a local variable so it’s accessible to the caller (last instruction in ‘fun’ and second last instruction in ‘start’. There was a simple example that covered return values earlier but now you return to again in the context of program scope.
  + **Second example**: 2fun\_return.py

def fun (num):

# Copy the value of 12 into a local variable

print(num)

num = num + num # Only local copy changed

print(num)

return(num) # Pass changes back to caller of function

def start():

num = 12

print(num) # OK within scope

num = fun(num) # Changes saved

print(num)

start()

* Students do, exercise #1
  + Modify the following program: exercise1\_return.py by implementing function doubleIt() that takes a numeric argument and returns a value that doubles that argument.
  + Students need to:
  + define the function that doubles the argument
  + Write the function call so that the doubled value for num can be displayed in start()
  + **Starting program**: exercise3\_return.py

def start():

num = int(input("Enter a number and I'll double it in returns: "))

print(num)

start()

* + **Solution**: exercise1\_solution.py

**def doubleIt(num):**

**num = num \* 2**

**return(num)**

def start():

num = int(input("Enter a number and I'll double it in returns: "))

**num = doubleIt(num)**

print(num)

start()

* **Third example, simple trace to determine output**: 3fun\_simple\_trace.py
* **Learning objective:** the order in which functions does not determine execution order, instead it’s the order of the calls to the function.

def a():

print("a")

c()

def b():

print("b")

def c():

print("c")

def start():

b()

a()

b()

c()

start()

* **Fourth example**, show them how to write Boolean functions: 4fun\_boolean\_function.py
  + What is a Boolean function (from “Switching Thoery” (Encyclopedia of Computer Science 2003) McLuskey, E. J. URL: <https://dl.acm.org/doi/10.5555/1074100.1074844>)
    - “In [mathematics](https://en.wikipedia.org/wiki/Mathematics), a **Boolean function** is a [function](https://en.wikipedia.org/wiki/Function_(mathematics)) whose [arguments](https://en.wikipedia.org/wiki/Argument_of_a_function) and result assume values from a two-element set (usually {true, false}, {0,1} or {-1,1}).”
    - Example: takes a numeric parameter and returns true or false depending if the number is zero or not.

def isZero(num):

if (num == 0):

return(True)

else:

return(False)

* **Students do, exercise #2** 
  + - Write a function ‘divide’ that takes the two numbers as parameters numerator, denominator and will divide the first number by the second only if the second is not zero. (The check if the denominator is zero should be done by the Boolean function just taught in tutorial). If the denominator is zero then the function should return the error code = -1. Test with: 1) non zero denominator 2) zero denominator

# Starting code: the function you just went over

def isZero(num):

if (num == 0):

return(True)

else:

return(False)

* + **Solution**: exercise4\_boolean\_function\_solution.py

def divide(numerator,denominator):

result = -1

**if (isZero(denominator) == False):**

result = numerator / denominator

else:

result = -1

return(result)

def start():

**num = divide(2,3)**

print(num)

**num = divide(2,0)**

print(num)

start()

* Parameter passing: pass by value (new learning concept: names are irrelevant with parameters because the actual argument and the formal parameters are two different memory locations, don’t use these terms with students just show that the caller has local copies as does the function being called)
* **Fifth example**: 5fun\_rudamentary\_parameter\_passing\_globals.py
  + Learning objectives:
    - Parameter passing: pass by value (names are irrelevant with parameters because the actual argument and the formal parameters are two different memory locations, don’t use these terms with students just show that the caller has local copies as does the function being called).
    - Scope: locals shadowing (hiding) globals.

x = 1

y = 2

# Local overrides global scope

def fun1():

x = 10

y = 20

print(x,y)

# Provides write access to one global

def fun2():

global x

x = 100

print(x,y)

# Proof: parameters are just a type of local

def fun3(num1,num2):

num1 = 1000

num2 = 2000

print(x,y,num1,num2)

# No locals, access global memory space

def start():

print(x,y)

fun1()

fun2()

print(x,y)

fun3(x,y)

print(x,y)

start()

**Output**

1 2

10 20

100 2

100 2

100 2 1000 2000

100 2

* + **Sixth example**: 6fun\_intermediate\_complexity\_parameter\_passing.py
  + **Learning objective**: reinforcing parameter passing and scope by including several different cases.

def fun1(num1,num2): # Same name but different locals

print("num1=%d\t num2=%d" %(num1,num2))

def fun2(x,y): # Different name but still just two local copies

print("x=%d\t y=%d" %(x,y))

# Num1 goes into num2, num2 goes into num1

def fun3(num2,num1): # Again two locals but transposed names

print("num1=%d\t num2=%d" %(num1,num2))

def start():

num1 = 1

num2 = 2

print("num1=%d\t num2=%d" %(num1,num2))

fun1(num1,num2)

fun2(num1,num2)

fun3(num1,num2)

start()

* **Seventh example:** 7complex\_trace.py
  + **Learning objectives:** (All functional decomposition concepts put together in one trace, also illustrates how poor style can make a program more complex to trace).

def end(): **#Poor maming**

print("Is", end=" ")

x = 12 **#Poor maming, global variable**

end()**#Haphard ordering**

print("this", end=" - ") **#Statements outside function body**

def d():

x = 22 **#Creates local that shadows global**

x = y **#Access global (y) to put into local (x)**

return(x)

def a():

pass

y = 23

def c():

**#Two locals shadow globals**

x = 21

y = 888

print(x,y)

def b():

print("start?")

def e(a,b,c): **#Poor naming: actual args vs paras don’t match**

global x **#Allows global to be changed in function (write)**

print(a,b,c) **#Copied global data into 3 locals**

a = 7

b = -13

c = 888

x = b **#This is what actually changes the global**

print(a,b,c)

return(a)

def start():

global y **#Allows global to be changed in function (write)**

print("the", end =" ")

b()

print(x,y) **#Access globals**

y = 19 **#Change global**

c()

print(x,y)

d() **#Return value lost**

y = d() **#Assignment changes global**

z = x

y = e(x,y,z) **#Globals as well as locals can be function args**

print(x,y,z)

start()