Introduction to Programming Using Python – Homework Exercises

# Introduction

These homework exercises are to help the student judge their progress in the subject. Do not fret if you have difficulties. Do not assume if they are easy that you shall not learn some interesting and useful items.

## Please remember

* Homework is for your benefit / fun.
* Do only as much / or as little as you can.
* Contact me with problems/successes/questions.
* Have fun!

## If you have problems:

* Think how you might solve the problem by hand.
* Try simplifying the task, in hope of adding to your solution as time permits.
* Ask the instructor for help. email?

## If the solution appears too easy:

* Consider if you have completely solved the problem. Error checking?
* Have you thoroughly tested your solution?
* Have you automated (written a program) to validate your solution?

## Please feel encouraged to email your solution and any other comments or questions to the instructor.

## Name you program files with short descriptive names e.g., hello\_world.py. All lowercase names are usually easiest to type and remember. Please test run your program(s), storing the output(s) in """ strings at the bottom of the source file or separate files named *program\_file\_without\_extension*.out *e.g,* hello\_world.out for program file named hello\_world.py.

Running and saving your program's output can be easily done by:

1. Starting IDLE, if not running
2. Open you program, if not present (**File**🡪**Recent** **Files**🡪*your file*
3. Running you program till done (**Run**🡪**e.g., Module** **(F5)**)
4. Save IDLE shell (File🡪**Save As**

**Save as Type:** All Types (\*.\*)

**File:name:** *program\_name***.out**

**SAVE**

)

Since you saved file is the most recently saved text file, you can look and edit this output by doing (**File**🡪**Recent Files->***your file***)**

## Our solution files / alternatives are mostly in folders …Introduction…/homework/Class\_.../

Our solutions for our twenty-question project are in …Introduction…/exercises/twenty\_question\_dev/iteration\_1.py, iteration\_2.py…

Please be encouraged to look at them if:

* You get stuck (Your always welcome to ask me for help via email/class)
* You have completed your solution and want to compare
* Are otherwise curious

If time permits, one gets the most out trying their own solution first.

# Class 1: Introduction - Getting Going

## Everything about Python we saw today

Make a list of all the topics we covered today. You don't need to be an expert in the topic. You need not be complete. A prize to the one who has the most items.

A start:

|  |  |  |
| --- | --- | --- |
| Topic | Example | Use |
| A program is set of instructions | print("Hello World") | Task process |
| Arithmetic operators | print(1+2-3\*4/5) | Calculation |
| Variables | a, max, customer\_name | Store values for later use |
|  |  |  |

## Hello world program (Our solutions: exerceses/introduction/hello\_world.py,…)

* Create and run a new file **hello\_world.py** - nothing new – just practice.
* Create and run a slightly different file **goodbye\_world.py**, using your first program as a starting point.

## Everything Program

Create a program file named "everything\_python\_1.py" which contains at least one example of every item you listed in 1. If any of your items intentionally cause the program to stop, write each of these as separate program files named "everything\_1\_err\_type" where type is tag describing the error type, e.g., "syntax" for syntax error.

You may combine items within one statement, but please indicate or comment items when used first time. Prizes given for: most items used, smallest program with most items

## Twenty questions Class Project Progress

It's a marathon (short 😊), not a sprint.

### Review / redo iteration 1 we did in class.

Iteration 1 – Loop forever, prompting for guess, accepting guess, printing guess

### Write and test Iteration 2. Remember start by saving a copy of iteration1.py as a new file named iteration2.py.

Iteration 2 – ADDING to Iteration 1 code: Set target value, Quit loop if number entered number equals target

## MY\_ATM – the beginnings of an ATM machine program

An ATM or Automatic Teller Machine facilitates the depositing and dispersal of money. Even without the mechanical aspects the programming involved can be very complex. Our program here will begin with just concerning the accepting and dispersing money. We will just consider the single individual having already logged on and verified. To further simplify the demands on the program only deposits or withdrawals will be accepted.

Program: File name: my\_atm.py

Operation:

1. Initial balance is 0.
2. Pressing the ENTER key with no amount will exit and display the balance
3. Positive number will withdraw that amount, IF sufficient balance, and display the new balance
4. Negative number will deposit that amount, and display the new balance.

Example **my\_atm.py** output:

Withdrawal AMT:50

Insufficient Funds for withdrawal

Balance: 0.0 request: 50.0

Withdrawal AMT:-100

Deposit: 100.0

New Balance: 100.0

Withdrawal AMT:75

Withdrawal: 75.0

New balance: 25.0

Withdrawal AMT:5

Withdrawal: 5.0

New balance: 20.0

Withdrawal AMT:-10

Deposit: 10.0

New Balance: 30.0

Withdrawal AMT:

Bye - Have a good day

# Class 2: Ideas, Tools, Functions

Note that the function is probably the most powerful tool in programming.

## Twenty questions Class Project Progress – Iteration 3

It's a marathon (short 😊), not a sprint.

Iteration 3: Say if guess is greater, less or equal

## Print with optional prefix – pprint.py

It is often useful to have a "print" function which acts like the traditional python print function but does something "special" for us, for example, precede the printed text with an optional prefix. Write and test a function:

pprint(arg, end=None, sep=None, prefix=None)

pprint acts like the standard print function, with one arg, but precedes the standard output with the value of prefix, if present.

Sample **pprint.py** output:

>>>

= RESTART: C:/Users/raysm/workspace/python/IntroductionToProgramming/homework/Class\_2\_\_Ideas\_Tools\_Functions/pprint/pprint.py

Test pprint

Testing no prefix: arg: Our String

Our String

Testing with prefix: arg: Our String prefix: PREFIX:

PREFIX:Our String

Testing with variable prefix:

Increasing prefix

=Increasing prefix

==Increasing prefix

===Increasing prefix

====Increasing prefix

=====Increasing prefix

======Increasing prefix

>>>

## Simple polygons, using turtle – polygons.py

Given the module turtle has rather basic operations such forward, right, one might like to add more elaborate figures to facilitate the creation of larger drawings. In this exercise, create and test the following function to create a rectangle:

rectangle(x=0, y=0, height=100, width=100, color="red")

which draws a red rectangle, starting at the coordinates 0,0 with the height of 100 and the width of 100.

Sample polygons.py output:

>>>

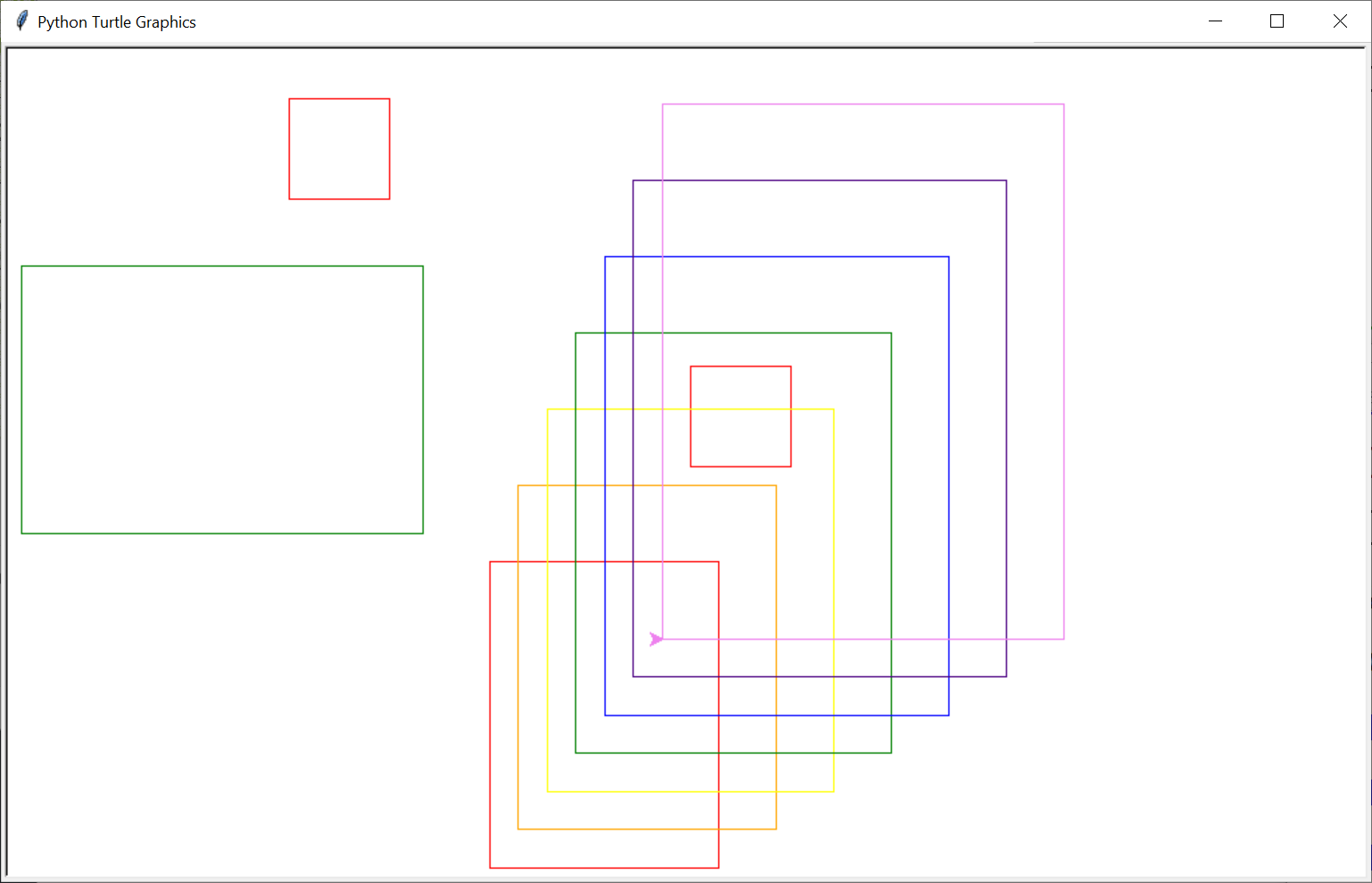
= RESTART: C:\Users\raysm\workspace\python\IntroductionToProgramming\homework\Class\_2\_\_Ideas\_Tools\_Functions\polygons\polygons.py

Simple Test - default rectangle

Simple Test - default size

Simple Test - one rectangle

Testing rectangle

Sample polygons.py Turtle Graphics screen: 

## More print keyword exercise - …homework/…/factors/…

Redo / extend factors work:

## Exercise 2: – Print factors – factors\_v1.py

## Omit 1, and number itself as factors - factors\_v2.py

## Only print numbers with at least one factor other than 1 or itself - factors\_v3.py

## Ask from, to - factors\_from\_to.py

## Salary Calculator

Sometime it may be nice to convert salary between yearly, monthly, weekly, and hourly. This program should convert salary in one pay period to the equivalent pay in all pay periods.

Salary Calculator

Loop

Ask salary pay period: Year, Month, Week, Hour

Ask salary, for this pay period:

Print salary yearly, monthly, weekly, hourly pay rates.

Sample **my\_salary.py** output:

>>>

= RESTART: C:\Users\raysm\workspace\python\IntroductionToProgramming\homework\Class\_2\_\_Ideas\_Tools\_Functions\budget\_calculator\my\_salary.py

Salary Calculator

supporting pay periods of:

year

month

week

hour

Enter BYE to quit

Enter pay period[year]: year

Enter pay amount per year: 65000

Salary: yr: 65,000.00 mth: 5,416.67 wk: 1,300.00 hr: 32.50

Enter pay period[year]: month

Enter pay amount per month: 3000

Salary: yr: 36,000.00 mth: 3,000.00 wk: 720.00 hr: 18.00

Enter pay period[month]: 2000

Sorry - we don't currently support pay periods of 2000

Enter pay period[month]: week

Enter pay amount per week: 1000

Salary: yr: 50,000.00 mth: 4,166.67 wk: 1,000.00 hr: 25.00

Enter pay period[week]: bye

Good Bye

>>>

## Budget Calculator (Not so easy)

Sometimes it may be useful to see how budget goals (e.g., save 200/year reflect on other time periods). This exercise is to build a program which takes an amount / percentage per time period (%, year, month, week) reflects on the other time periods. **Hint**: Start with my\_salary.py

## Ask salary pay period: Year, Month, Week

## Ask salary, per pay period:

## Loop

## Ask budget(save) period: Percent, Year, Month, Week

* + 1. Ask budget(save) amount: per period or percentage
    2. Print percentage, yearly amount, monthly amount, weekly amount

Sample **my\_budget.py** Output:

>>>

= RESTART: C:/Users/raysm/workspace/python/IntroductionToProgramming/homework/Class\_2\_\_Ideas\_Tools\_Functions/budget\_calculator/my\_budget.py

Budget Calculator

supporting pay periods of:

year

month

week

hour

Enter BYE to quit level

Enter pay period[year]:

Enter pay amount per year:

Assuming amt: 1000.

Salary: yr: 1,000.00 mth: 83.33 wk: 20.00 hr: 0.50

Enter Budget period or %[%]

Assuming: %

Enter Budget percent:[10]:

Budget: 10.0% yr: 100.00 mth: 8.33 wk: 2.00 hr: 0.05

Enter Budget period or %[%]month

Enter monthly Budget[0]:8.333

Budget: 10.0% yr: 100.00 mth: 8.33 wk: 2.00 hr: 0.05

Enter Budget period or %[month]hour

Enter hourly Budget[8.333]:.1

Budget: 20.0% yr: 200.00 mth: 16.67 wk: 4.00 hr: 0.10

Enter Budget period or %[hour]%

Enter Budget percent:[10]:20

Budget: 20.0% yr: 200.00 mth: 16.67 wk: 4.00 hr: 0.10

Enter Budget period or %[%]bye

Enter pay period[year]: bye

Good Bye

>>>

## Multiplication Table

In my early school days, when attempting to learn and reinforce my arithmetic skills I would sometimes create multiplication tables which would display the product of two numbers from 1 to 12, e.g., 7\*8 giving 56. The table would have the number of one operand across the top and the other operand displayed down the left side. This exercise will be the creation of such a table.

The following is a run output for a simple implementation:

>>>

= RESTART: C:\Users\raysm\workspace\python\IntroductionToProgramming\exercises\times\_tables\times\_tables\_simple\_for.py

Enter times table length: 6

1 2 3 4 5 6

2 4 6 8 10 12

3 6 9 12 15 18

4 8 12 16 20 24

5 10 15 20 25 30

6 12 18 24 30 36

>>>

= RESTART: C

One might start with this goal and add:

## Formatting the products to be right aligned

## Legends on top and left which display the operands

## Support the specification of maximum and minimum for each operand.

The following is a run output with the above extensions:

= RESTART: C:/Users/raysm/workspace/python/IntroductionToProgramming/exercises/mult\_table/mult\_table\_formated\_wleg.py

nmin[1]: 999999

nmax[1000004]: 1000001

mmin[999999]:

mmax[1000001]:

table 999999 to 1000001 by 999999 to 1000001

999999 1000000 1000001

+------------- ------------- -------------

999999| 999998000001 999999000000 999999999999

1000000| 999999000000 1000000000000 1000001000000

1000001| 999999999999 1000001000000 1000002000001

# Class 3: Functions – Why and How

## Twenty questions Class Project Progress – Iteration 4

It's a marathon (short 😊), not a sprint.

Iteration 4: Set target to random number

Don't forget "import random". Take a look at Python module random:

IDLE 🡪 Help 🡪Python Docs 🡪modules->r->random

## Import random

use

target = **random.randint**(*1*, 25)

## from random import randint

target = **randint**(1,25)

## Playing cards - deriving card functions from friends\_family

Developing card playing games can be similar to our friends\_family set of files. In this problem the task is to develop a group of functions that facilitate the manipulation of cards. We are going to consider playing cards as text strings of the following format:

<Suit letter: S for Spades, H for Hearts, D for Diamonds, or C for Clubs>

followed by the ":" character

followed by the <rank: A for Ace, K for King, Q for Queen, J for Jack, or 1, 2, 3, … 9 for card number>

Examples: S:A for Ace of Spades, C:2 for duce of Spades, S:Q for queen of Spades

Please develop the following functions.

def list\_hand(prefix=None):

""" list cards in hand

:prefix: optional prefix for id

default: no prefix

"""

def add\_one\_card(card):

""" Adds one card to our list

:card: card's name

"""

def add\_cards(\*cards):

""" Add zero or more cards

:\*cards: zero or more card names

"""

def has\_card(ck\_card):

""" Check if card is in my\_hand

:ck\_card: name of card

:returns: True if new\_card is in my\_hand

"""

You are strongly encouraged to use the files exercises/functions/friends\_mod.py and friends\_4.py as a starting point – I did for the solutions.

Sample output:

>>>

= RESTART: C:/Users/raysm/workspace/python/IntroductionToProgramming/exercises/functions/playing\_cards/test\_cards\_mod.py

add\_cards( S:A S:K S:Q S:J S:10 )

add\_one\_card(S:A)

my\_hand: S:A

add\_one\_card(S:K)

my\_hand: S:A, S:K

add\_one\_card(S:Q)

my\_hand: S:A, S:K, S:Q

add\_one\_card(S:J)

my\_hand: S:A, S:K, S:Q, S:J

add\_one\_card(S:10)

my\_hand: S:A, S:K, S:Q, S:J, S:10

my\_hand: S:A, S:K, S:Q, S:J, S:10

S:A is in hand

S:K is in hand

S:Q is in hand

S:J is in hand

S:10 is in hand

H:A is NOT in hand

D:K is NOT in hand

C:Q is NOT in hand

>>>

# Class 4: Classes and More

## Twenty questions Class Project Progress – Iteration 5

It's a marathon (short 😊), not a sprint.

Iteration 5: Say goals, rules before start, including "a number between…"

Don't forget about multi-line (""") strings, and f-strings with inserted values

## Dictionaries / Commands

In **exercises/dictionarys/motion.py** seen that python dictionaries can be used in building command-based programs/games. In this exercise we would like to extend this program/game. Please add and exercise the following additions:

### Setup start, goal, test for goal

It would be a great idea to test after adding each of the following.

#### Start the motion location (irow, icol) at a random point within the boundaries (0-board\_width; 0-board\_height) displayed ("S").

#### Place the goal (displayed as "G") at a random place within the board boundaries but not on the start.

#### After each move, test if at goal and announce a win if found.

# Class 5: Files – Data that stays around

## Twenty questions Class Project Progress – Iteration 6

It's a marathon (short 😊), not a sprint.

Iteration 6: Ask player if they want another game – play multiple times

# Class 6: More

## Twenty questions Class Project Progress – Iteration 7

It's a marathon (short 😊), not a sprint.

Iteration 6: Handle typos / illegal numbers

# Class 7: Still More – Where to now?