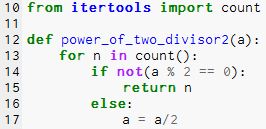
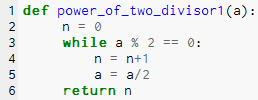
Python course work 1

1.a) Using the given functions power\_of\_two\_divisor1 and power\_of\_two\_divisor2 given below:

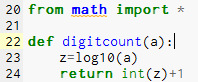




Testing power\_of\_two\_divisor1 by first running both functions in the command window and then typing power\_of\_two\_divisor1(7) gives 0 which I correct. Then testing it again but instead of 7, testing it with 14, 28. This gives outputs of 1 and 2 which is again correct since 2 divides 14 once and 28 twice.

Similarly testing power\_of\_two\_divisor2 gives the same results with the same inputs, so power\_of\_two\_divisor2 also works for smallish positive integers.

b) Using the following code in the editor and running it gives a function digitcount which returns the number of digits a number has.



So testing on the following values in the command window 8, 100, 232323 gives 1,3,6 which is clearly correct and thus working.

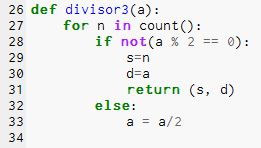
c) Again using power of 2 functions on some large random integers (about 200 and 300 digits respectively) which are given below (and I will be using again):

t=747578545748645630752647654207654072540650726540756420754260547265407654076542073607546207546507265065407657056076225076507456045620753635076206532754069765475467664597260265075462705462907630765407657063207637050764576250476054

w=8514086510615704657941654076541765147122242442424424242442424262626282882628628628628286286268267879459875498568758548598547857852235786868686868628268268268262862898795417689752469527594765906754025640754205672265387504765690327506572606590875698754690865454986538653986549865485320856626262424424244

The functions of two both seemed to cope with the large integers easily enough taking very little time to give an output.

d) Again here is the code in the editor for the function divisor3 (a modified code for power\_of\_two\_divisor2) given below which takes as an input a positive integer and outputs the number of times 2 divides it and an integer which is left after it cannot be divided by 2 anymore.



e)Again testing the function in the command window with the following inputs 8, w and t gives the following respective outputs:

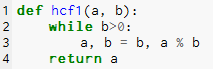
(3, 1)

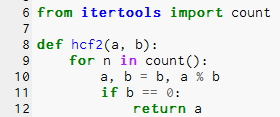
(2,2128521627653926164485413519135441286780560610606106060610606065656570720657157157157071571567066969864968874642189637149636964463058946717171717157067067067065715724698854422438117381898691476688506410188551418066346876191422581876643151647718924688672716363746634663496637466371330214156565606106061L)

(1,373789272874322815376323827103827036270325363270378210377130273632703827038271036803773103773253632532703828528038112538253728022810376817538103266377034882737733832298630132537731352731453815382703828531603818525382288125238027L)

All of which are correct. And when being tested in all cases the answer was given without delay, so seemed to be able to cope with big and small integers without loss of information.

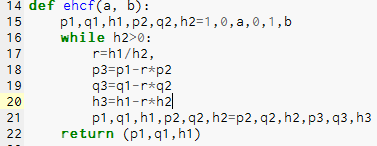
2.a) Testing the functions hcf1 and hcf2(given below) in the command window(running the program first) on the intergers (67,34), (124,248), gives the following results from both functions respectively of 1 and 124 which are both correct. So the functions work on smallish integers.





Then testing both the hcf functions on larger values e.g (w ,t ) gives 2L without any apparent change in speed of calculation.

b) The following code below gives the function ehcf.



Testing this function by running it and then typing ehcf(8,14) command window gives (2,-1,2) which is correct since 2\*8-1\*14 is 2.

Then testing it on (1000000000000000000000000000000000,99999999999999999999999999999999) gives (10000000000000000000000000000000L, -100000000000000000000000000000001L, 1L) which is also correct.

Then testing it on (t,w) gives (1028996915100817433449477812962586498181652985909515867026323174772968192024209935133501790543772672689042778786481392576624986754609680244349662201676152347240531827745083213975949951717665323413809897019690748930261593749397350533938624793378503031995392475401309143662160654434317783127237022798933L, 90350974988540744062390391200138444730752446985909652404972023763679796723139461635726644534547895558775967805263854609784649895450763724368237631353915023684673719090067168464866650564249704618123874059263264850349214978870436L, 2L) which is also correct, the function appears to work with large integers without any loss of speed.

c) To calculate p\*2^36+d\*3^10=1 you can use the ehcf function using an input of ehcf(2\*\*36,3\*\*10) in the command window after running the program. This gives an output of (15904L, -18508604007L, 1L), thus a value of d is -18508604007, this works because the hcf of the two numbers is 1.

Thus d=-18508604007

d) A way of finding a positive d is by just adding n(as a result of how modular arithmetic works).

Thus the new d1=d+n=-18508604007+2\*\*36=50210872729

So d=50210872729

3.a) The following is the code for the desired function and is called miller, it does however relies on a slightly altered divisor3 function from question 1.

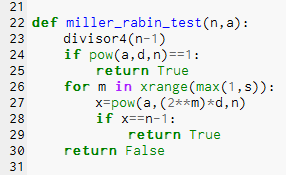


So testing it by first running the program and then typing in the command window miller(13,7) gives an output of [5, 12] which is correct since 12 is -1 mod 13 which according to the lemma means its prime thus this case verifies the lemma as 13 is indeed prime .

And then testing in the same way with inputs (13,5), (13,2), gives the respective outputs [8, 12], [8, 12], as 12 is in all the outputs, this verifies the lemma further.

And then test in the same way again with inputs (561,235), and (561,235) gives the respective outputs [364, 100, 463, 67] and [485, 166, 67, 1], which both suggest that 561 is not prime, since 560 is not in the list, so again this supports the lemma.

b) The code below is for the function called miller\_rabin\_test, it again relies on the altered function divisor4 given above.



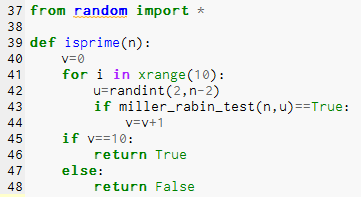
Again I tested the function by first running the above program (which includes divisor4 code) and then typing miller\_rabin\_test(13,5) it gives the following output of True which is correct as 13 is prime.

And then testing in the same way with inputs (1468,48), (t,567),(67,3) gives the respective outputs of False, False and True, all of which are correct.

And then test in the same way again with inputs (561,235), and (561,511) gives the respective outputs False and True. The reason for this is 561 can be a liar, it is not prime as it can be divided by 17. The reason it fails is because passes the first part of the lemma(a\*\*d=-1 mod(n)). However if you test it for most of its bases with the following code given below in command window, you discover it only fails to approximately 8 bases so very rarely will the test function actually fail. So most of the time the output will be false which is correct.



c) The code below is for the function isprime, which takes an input of an integer and outputs True or False depending on if it is prime.

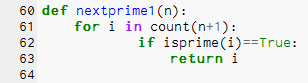


So I tested it by first running the program which includes divisor4 and miller\_rabin\_test and then typing in the command window isprime(13). This gives an output of True, which is correct since 13 is prime.

I also tested it on the following numbers 14,20,4392318012,t,w all of which return false which is correct.

I also tested it on the following numbers 17,19,67 and the following from the online website given in question sheet 5915587277, 2074722246773485207821695222107608587480996474721117292752992589912196684750549658310084416732550077. All of which returned true which is correct since all are prime.

d) The code below is for the function nextprime1 and is in same file as isprime, miller\_rabin\_test and divisor4.



Using the nextprime function with an input of 10\*\*100 gives;

10000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000267L

Which is the next prime number after a googol.