python details – DOES NOT COVER FUNCTIONS!!!

This is a sample of some of the things that you are responsible for – do not believe that if you know only the things on this test that they will get an A on any real test.

You must know everything that has been covered in lecture, in the text, in the online recitations and in the labs.

The purpose of this exercise to let you learn some of the things you do not know – so you can go out and learn them.

Note that this worksheet does not thoroughly test your problem-solving / programming ability but on the test you will required to write programs and/or short sections of code (code fragments)

1. What is the purpose of variables? to hold data in the program – to store information – to allow different parts of a program to communication information.

2. State the rules for valid identifiers (what’s an identifier)?
   Start with: letter or _ (strongly suggest that don’t use initial _’s in this course)
   Followed by letters, digits, _ (no spaces)
   Cannot be a reserved word.

3. What are the 7 arithmetic operators? (symbols and names) ** * / // % + -

4. State the rules regarding the types of the operands with arithmetic operators (what’s an operand?)
   must be numeric quantities to get numeric results,
   for str + means concatenate and * means repetition

5. State the rules about the value and type of arithmetic expressions (what’s an expression?)
   * / + - if operands are the same type, so is the result (or possibly “bigger” if value is larger)
   if operands are different types, result is of “larger” type.
   [the order of “size” is int → float]
   in python 2.6 / still give integer division if both operands are ints or longs
   // always give integer division

6. What is the type and value of 13 / 7 (or 13 // 7)? of 3 + 2 * 8 ?
   int, 1
   int, 19

7. What are precedence and associativity?
   precedence rules say which operators do their work before others (see previous question)
   associativity rules say whether operators work from L → R or R → L
8. Round off the value that variable q refers to.
   \[ q = \text{round}(q) \]

9. Multiply q by \( \pi \).
   \[ q *= \text{math.pi} \]

10. Define a variable that will refer to whether or not the user entered the number 5.
    
    ```python
    usernum = \text{raw_input}(" Enter a number ")
    got5 = usernum == 5
    ```

11. Write the word YES on the screen if the user did enter the number five – do this using your variable from the previous question.
    
    ```python
    if got5:
        \text{print} \ "\text{YES}"
    ```
    
    Do not write:
    ```python
    if got5 == True:  \# NO NO NO NO!!!
        \text{print} \ "\text{YES}"
    ```
    
    Or you should have written:
    ```python
    if (got5 == True) == True:  \# NO NO NO NO!!!
        \text{print} \ "\text{YES}"
    ```

12. What are the bool values? only True and False – 0 and 1 are NOT bool values.
13. Can numeric values coerce to bool values? (what’s coercion? what’s casting?)
   Coercion is forcing a value to another type. so is casting
   Give some examples of numeric to bool conversions.

   0  → false
   0.0 → false
   1  → true
   -1 → true
   3.154333 → true
   "hello" → true

14. List the six relational (comparison) operators < >= > <= == !=

15. Which of these six operators is a branching statement?
   NONE – each of these could be used in a test in either the if part or the elfi part
   (or in the while test for that matter but while is not a branching statement)

16. State the type rules for operands with the relational operators.
   Roughly the same as for arithmetic operators. Comparing an int and double causes the type of the int
   value to become double before the comparison is tested.
   Note the problem if both operands are double (may not be able to test for exactly equal)
   Is 1/3  0.3333333333333333333333333… Cannot accurately store decimal fractions.
   Also for string., use lexicographical sort ordering:
   >>> 'he'>='hello'
   False
   >>> 'jello'<'hello'
   False

17. What is the type of an expression involving a relational operator? bool (NOT int, NOT boolean)

18. What is the opposite of < ? >= (NOT >)

19. What is the type and value of 3 != 3 ? type: bool, value: False (NOT 0)

20. List the logical (boolean) operators not and or

21. State the type rules for operands to be used with the logical operators.
    Both operands must be bool or coercible to bool or using these doesn't make any sense

22. What is the type of an expression involving a logical operator? bool (NOT int)

23. What is the type of x in range( 5 )? bool
24. Is in a relational operator? no – it's the set inclusion operator
25. What is opposite of in? not in (one operator)

26. What is the type and value of ( 3<4 ) || ( 3 >= 8 )
    type: bool (not Boolean)
    value: True (NOT 1)
27. If \( r \) and \( t \) are floats, is \( r != t \) always going to produce the result you might expect? Why? No, values like 1/3 cannot be stored completely accurately so testing reals sometimes will not work.

28. What is short circuit evaluation?
For expressions involving \( \text{and} \), evaluation moves from left to right and \( \text{stops} \) at the first False subexpression
For expressions involving \( \text{or} \), evaluation moves from left to right and \( \text{stops} \) at the first True subexpression

29. What is precedence of \( \text{and} \) or \( \text{not} \)?
   \( \text{not} \) has higher precedence (and it's unary), then \( \text{and} \) then \( \text{or} \)

30. Consider the following with and without the parentheses?
   When does the system stop evaluation subexpressions?
   \(((3<4)\text{and}(8!=8))\text{or}(9>=2)\)
   With parentheses: ( ) and ( ) first. Short circuit evaluation forces the \( \text{and} \)'s left operand to be evaluated before it’s right operand. So the \( \text{or} \) must be evaluated and it stops evaluating after it determines that \( (3<4) \) is True. \( (8!=8) \) is \( \text{not} \) evaluated. Now \( \text{and} \) must evaluate \( 9>=2 \).
   Because \( \text{and} \)'s left operand, \( ((3<4)\text{or}(8!=8)) \), was True, it could use short circuit stop its evaluation.
   Without parentheses: \( 3<4 \) or \( 8!=8 \) and \( 9>=2 \)
   Precedence is in this order: \( < != > = \) before any \( \text{logicals} \) then any \( \text{and} \)'s before any \( \text{or} \)'s giving this implied parenthesized expression: \( (3<4)\text{or}((8!=8)\text{and}(9>=2)) \)
   Now the short circuit evaluation of the ( ) or ( ) expression stops after \( (3<4) \) is determined to be True. \( \text{No part of} \ ( (8!=8)\text{and}(9>=2)) \) is evaluated.

31. Why is \( ( 3 < 4 ) == \text{True} \) dumb (besides testing literals)?
   because a relation expression like \( 3<4 \) IS True (or False)
   bool values should be used as bool values, not tested against bool values.

This becomes clear when used in an if statement:

```
if (3 < 4) == True: ... dumb
if 3 < 4: ...          good
```

32. What is the value of \( x = y \)?

This is an assignment statement, not an expression.
   It does NOT have any value – it's a statement.
   This is like asking what is the value of a if-else statement – it does not have a value it is an activity..

33. What is the value \( \text{range}(3) \)?
   \[0,1,2\] NOT \[0,1,2,3\]

34. What are the possible values of \( \text{random.randint}(3) \)?
   0 or 1 or 2 or 3  The 3 IS included.

35. What is the value \( \text{range}(3,12,3) \)?
   \[3,6,9,]\] Does NOT include 12
36. There are four main kinds of statements. We group them as follows:

- statements that happen once and are done
- statements that control the execution of other statements
  - looping
  - branching
- statements that cause execution to jump to another place in the program
- exceptions (which are not part of this course)

You must know the 8 problem forms for looping and branching.

A statement is kind of like an “activity unit” in the execution sequence.

Below are all the python statements we have covered so far along with some things that are not statements.
Identify which of the categories listed above each statement falls into.
For the looping and branching, state the problem forms each is a solution to.
If it is not a statement, say so and explain.

Here we have answered the Which kind of statement is it? by grouping all the statements together.
(and all the non-statements)

- statements that happen once and are done
  - input statement
  - output statement
  - assignment statement (like \( x = 3 \) or \( y *= 7 \) or \( z = m \))
  - assignment statements are either variable definition or variable re-references
  - global const definition – it’s still a definition. non-const globals should not be used

- statements that control the execution of other statements
  - looping
    - for var in range(N) known or knowable number of times (N)
    - for var in sequence is is used to process the sequence of values
    - while unknown number of times but can test for "it's not the end"
  - branching
    - if do or don't do
    - if else do one of 1 must do 1
    - if elif elif … else more than 2, one must happen (else is default)
    - if elif elif … elif more than 2, one must happen last elif means a test is made
    (no switch in python)

- statements that cause execution to jump to another place in the program
  - return _____ used in functions to cause to give the value back to whomever called the function.
a function call can also be thought of as a jump to the code contained in the function def

- these are NOT statements
  - \( x + y \) – this is an expression. It has type and value but is not by itself a statement. If can be used in an output statement or an assignment statement, of course like \( z = x + y \)
  - else – does not exist without being part of an if-else
  - elif – does not exist without being part of an if-else

37. What type must the test be in any controlling statement? The test is the thing inside the ( ) in a controlling statement that is used to make the decision.

```
bool
```

38. What are the three looping statements in python and when do you use each?

```
while  for var in range(N) nad for var in sequence
```

These control any python statement including themselves

- for \( var \) in rang(N) is used to do something N times
- for \( var \) in sequence process a sequence of values
- while is typically used when we do not know when the loop should end but can write a test that will allow us to figure out when the loop should stop.

39. What are the branching statements in python and when do you use each?

```
if,  if-else,  if-elif-elif-else,  if-elif-elif,
```

```
if – do something or don’t do it
if-else – do exactly one of two possible things to do
if-elif-elif-else – chose one of many possibilities – exactly one will happen
if-elif-elif-elif – chose one of many possibilities – exactly one of the possible choices or none of them will happen
```

40. Why is this code wrong?

```
if n < 4:
    print “blah blah\n”
if n > 4:
    print “other blah blah\n”
if n == 4:
    print “yet more blah blah\n”
```

The programmer is not using the information gained by making tests.
If the first test is True, both of the following tests are absolutely known to be False and there is no need to test for these situations.
This sequence of if’s should logically be written:
if n < 4:
    print "blah blah\n
elif n > 4:
    print "other blah blah\n
else:  # ( n == 4 )
    print"yet more blah blah\n"

Notice the else has a comment showing what is True at that point

41. Write the code that asks the user for their birthdate (an int like 27) and prints YES on the screen if their birthdate is the same as yours. Question: what is supposed to happen if their birthdate is not the same as yours? Question: would there be any constants? Question what is a code fragment? On the test if we ask you to write a code fragment or show you a code fragment, do you know what is meant?

A code fragment is a few lines of code. It is NOT a complete program.

On the test if we ask you, like in this question, to write some code, unless we specifically state that you must write a complete program, we are only asking for a fragment. Only the lines of code that are needed to solve this specific small problem. For this problem you need to only write the variable definitions, prompt the user and get the input from the user and the if statement that controls the printing of YES. You are not being asked to write a complete program.

If, on a test you waste your time writing a complete program when we only ask for a code fragment, you will make a lower graded. We will not grade the whole thing – unless there is an error. You can only lose points for doing something that is not asked for.

MY_BDAY = 7

userBday = raw_input( "Please enter your birthdate: " )

if userBday == MY_BDAY:
    print "YES"

42. To thoroughly test your code from the previous question, how many runs of the program would you need to do and what would be the values you need to input when pretending to be the user? At least two: 27 so that YES appears and then some other number to make sure nothing happens. You would NOT need to test things like what happens if the user types the letter R for this problem. For this test, unless we very specifically state otherwise, you can assume that the user will not input the wrong kind of information – like hitting letter keys on the keyboard when we ask for numbers.

43. What is the output of this code?

NOTE: this is only a code fragment. If you say there is no output because there is not a complete program, you will have earned a ZERO for this question on a test. Assume the rest of the program is there and works. Note that you will not be allowed to type this code into .NET and run it on a test. No computers of any sort – including calculators – are allowed on the tests and final.

for q in range( 4 ):
    for s in range( q ):
        print str( q ) + " " + str( s )
44. What is the scope of variable q in the last question?

Local to the def this code if written in.

45. Describe or write the code for calculating an average of user inputs using the sentinel method
create a sentinel value as a const
tell the user to enter all the things to be averaged and to enter the sentinel value when done
get the first input
while he current input != the sentinel value:

    update the accumulator ( sum += current input )
    update the counter (counter += 1)
    get the next input

} AFTER the loop, calculate the average
You must check for division by zero
You may have to use cast one side of the division to float if the things to averaged are integer values
The counter must be an int (or long) type and the accumulator must be an int (or long) type if the
things to be added up are all integers

46. Describe or write the code for calculating an average of user inputs using the “ask how many”
method where the user is asked how many inputs there will be.

ask the user how many things will be entered
get the that number
for loopVar in range( that number )
{
    get an input
    update the accumulator ( sum += current input )
    // we don’t use a counter
}
AFTER the loop, calculate the average
You must check for division by zero
You may have to use cast on of the sides of the division to float if the things to averaged are integer values
(The number of things to be entered must be an int or long type and the accumulator must be an int or
long type if the things to be added up are all integers.)

47. Describe or write an input validation loop
The general pattern is the same for averages and for passwords:
get the first input
while it’s not valid: # this is the test for not valid

  complain
  ask for another input
  get the next input

48. What are all the uses of variables that you know

input variables (used to store a user’s input)
calculated variables (used to store the results of a calculation)
known data (should be a const and consts must be initialized)

counters (must be initialized to 0, must be += 1 inside the loop)
accumulators (must be initialized to 0, must be += current inside the loop)

for loop variables (scope concept SHOULD be only inside the for statement)

49. Last Question: What topics, before functions, was not covered in this review?

Answer: this document has nothing to do with function!!!!