2. (i) True
   (ii) True
   (iii) False
   (iv) False

3. (i) False
   (ii) False
   (iii) True
   (iv) e.g. !P||P && !Q||Q (your example may differ)

4.

Algorithm: modulus(x) Requires: one real number x Returns: the modulus of number x
1: if x>=0 2: return x 3: else 4: return -x 5: endif

5.

Algorithm: reverse(n) Requires: 3-digit integer n Returns: integer with reversed digits of n
1: if n%10==0 2: return -1 3: else 4: let a = ------- 5: let b = ------- 6: let c = n%10 7: return 100*c+10*b+a 8: endif

6.

Algorithm: NOT(P) Requires: one Boolean variable P Returns: the negation of P: True/False
Write body part on your own

7.

```
Algorithm: AND(P, Q)
Requires: two Boolean variables P and Q
Returns: the compounded result AND(P,Q)
1:  if P
2:      if Q
3:          return True
4:      else
5:          return False
6:      endif
7:  else
8:      return--------
9:  endif
```

8.

```
Algorithm: OR(P, Q)
Requires: two Boolean variables P and Q
Returns: the compounded result OR(P,Q)
1:  if P
2:      return ----------
3:  elseif Q
4:      return True
5:  else
6:      return False
7:  endif
```

9. (i) $y = 2x - 1$

(ii) 49

(iii)

```
Algorithm: findX(y)
Requires: one (odd) integer variable y from Set Y
Returns: the value in Set X which results in y
1:  return (y+1)/2
```

(iv) The rule we are following to find x is the inverse function of $y = 2x - 1$.

10.

```
Algorithm: isTriangle(a,b,c)
Requires: three real numbers a, b, and c
Returns: True if we can form a triangle; False otherwise
1:  if a<b+c && b<c+a && c<a+b
2:      return True
3:  else
4:      return False
5:  endif
```

Note: Lines 1-5 can be replaced by the following statement:

```
1:     return (a<(b+c))&&(b<(a+c))&&(c<(a+b))
```

(Parentheses helps to understand statement.)

11.

P	Q	NAND(P, Q)
True	True	False
True	False	True
False	True	True
False	False	True

Algorithm: NAND(P, Q) Requires: two Boolean variables P and Q Returns: NOT-AND gate of P and Q
1: if P 2: if Q 3: return ---------- 4: else 5: return ---------- 6: endif 7: else 8: return---------- 9: endif

Note: (i) Correction: this logical gate should be called **NOT-AND** instead of Negative-AND. Please make this correction in the original question.

(ii) NOT-AND (NAND) can be described as !P || !Q. So Lines 1-9 can be replaced by following statement:

```
1:     return !P || !Q
```

(iii) Negative-AND can be described as !P && !Q.

12.

Algorithm: XOR(P, Q) Requires: two Boolean variables P and Q Returns: exclusive OR gate of P and Q
1: if P 2: if Q 3: return False 4: else 5: return True 6: endif 7: else 8: if Q 9: return True 10: else 11: return False 12: endif 13: endif

Note: Lines 1-13 can be replaced/simplified as

```
1:    return P&&!Q || !P&&Q
```

Or if you could notice the result actually depends on whether P, Q are the same, following statement also works:

```
1:    return P!=Q
```

13.

Algorithm: taxCalc(x) Requires: one positive integer x Returns: annual income tax
1: if x > 90000 2: return -1 3: elseif x>65000 4: return 10000*0.05+15000*0.075 + 20000*0.11+20000*0.15+(x-65000)*0.175 5: elseif x>45000 6: return 10000*0.05+15000*0.075 + 20000*0.11+(x-45000)*0.15 7: elseif x>25000 8: return 10000*0.05+15000*0.075 + (x-25000)*0.11 9: elseif x>10000 10: return 10000*0.05+(x-10000)*0.075 11: else 12: return ---------- 13: endif

14.

Algorithm: whatDay(n) Requires: one integer number n ranging from 1 to 30 Returns: one integer representing the week day (ranging from 1 to 7)
1: let x = n%7 1: if x==1 2: return 4 // Thursday 3: elseif x==2 4: return 5 // Friday 5: elseif x==3 6: return 6 // Saturday 7: elseif x==4 8: return 7 // Sunday 9: elseif x==5 10: return 1 // Monday 11: elseif x==6 12: return 2 // Tuesday 13: else // x==0 14: return 3 // Wednesday 15: endif

Comments:

i. All solutions presented here are "reference solutions", meaning that your algorithms could be different from the above, provided that they are correct.

ii. Tracing your own algorithms can help you check whether they are correct.