**1.** d. key

2. b. False

3. b. False

4. b. Value None is returned

5. b. tuple containing the pair of last item of the dictionary

6. b. tupple containing the pair of last item of the dictionary

7. a. items() and d. keys()

8. b. {} curly brackets

9. dict = {

'color' : 'yellow',

'fruit': 'banana'

}

dict['color']

dict['banana']

dict['color'] == 'yellow'

10.

**11.** X[0:3] == ['Feb', 'Apr', 'Mar']

X[2:8] == ['Mar', 'May', 'Jun', 'Jul', 'Aug', 'Jan']

X[4:9] == ['Jun', 'Jul', 'Aug', 'Jan']

X[1:7:2] == ['Apr', 'May', 'Jul']

X[-1:-7] == []

X[-7:7] == ['Apr', 'Mar', 'May', 'Jun', 'Jul', 'Aug']

X[-1:-8:-2] == ['Jan', 'Jul', 'May', 'Apr']

X[:4] == ['Feb', 'Apr', 'Mar', 'May']

**12.** 3) .remove(9)

13. 1) \*

14. 1) .upper()

**15.** <class 'bool'>

<class 'str'>

<class 'int'>

<class 'float'>

**16.** 2) p and R2

**17.** 4) p.x()

18. X=4 , Y= 2

print(X % Y) == 0

print(X / Y) == 2.0

print(X // Y) == 2

print(Y % X) == 2

18. missing from assignment

19. x = [[4, 1, 1], [5, 9, 0]]

for i in range(0, len(x)):

for j in range(0, len(x[0])):

list = x[i][j]

print(list)

Output: 4

1

1

5

9

0

-----------

x = [[4, 1, 1], [5, 9, 0]]

for j in range(0, len(x[0])):

for i in range(0, len(x)):

list = x[i][j]

print(list, end='\t')

print()

Output: 4 5

1 9

1 0

--------------

x = [[4, 1, 1], [5, 9, 0]]

x = [[4, 1, 1], [5, 9, 0]]

for i in range(0, len(x)):

for j in range(0, len(x[0])):

list = x[i][j]

print(list, end = '\t')

Output: 4 1 1 5 9 0

---------

x = [[4, 1, 1], [5, 9, 0]]

x = [[4, 1, 1], [5, 9, 0]]

for i in range(0, len(x)):

for j in range(0, len(x[0])):

list = x[i][j]

print(list, end = '\t')

print()

Output: 4 1 1

5 9 0

20. q = [10.62, 16.14, 6.45, 17.11]

for j, z in enumerate(q) :

print( 'Item ' + str( j ) + ' - ', str ( z ))

Output: Item 0 – 10.62

Item 1 – 16.14

Item 2 – 6.45

Item 3 – 17.11

Ans: 3) j

21. b) The keys of a dictionary can be accessed using values

22. a. {0: 1, 1: 1, 2: 1}

23.  def increment\_score ( bonus , score , points = 1 ) == True  
 def increment\_score ( bonus = True , score , points = 1 ) == False

 Once a parameter is defined with default value, any parameter to the right must also be defined with default values == True

24. def avg(x, y, z = 50):

adding = x + y + z

avg\_value = adding / 3

return avg\_value

y = avg (x = 5, y = 9, z = 20)

print(y)

Output:

11.333333333333334

**25. What will be output? Describe it with reason and logic behind.**

**def avg(\*opt\_values, name):**

**avg\_value = sum(opt\_values) / len(opt\_values)**

**print('Name: ' + name + '\n' + 'Marks: ' + str(avg\_value))**

**avg(5, 9, 20, 34, 87, 112, name = 'Ali')**

Output:

Name: Ali

Marks: 44.5

**Ans:** We are defining a function *avg* that accepts positional arguments through positional parameter **opt\_values** and keyword-only argument through named parameter **name**. Since we want to allow arbitrary number of arguments to pass into the function we use an asterisk (\*) before the parameter name **opt\_values** to denote this kind of argument.

Keyword-only arguments are function arguments which can only be specified using the keyword syntax, meaning they cannot be specified positionally. When we call a function with some values, these values get assigned to the arguments according to their position. Python allows functions to be called using keyword arguments. When we call functions in this way, the order (position) of the arguments can be changed.

We call the function with arbitrary number of positional arguments **opt\_values** and the keyword-only argument **name**, summing and averaging the **opt\_values,** we print them. We print *avg\_value* in the data type string because we can only concatenate string, not float, to string.

The argument **name** comes after \***opt\_values** , which means it can only be specified as a keyword argument. If we try to specify it positionally we’ll get an error:

def avg(\*opt\_values, name):

avg\_value = sum(opt\_values) / len(opt\_values)

print('Name: ' + name + '\n' + 'Marks: ' + str(avg\_value))

avg(name = 'Ali', 5, 9, 20, 34, 87, 112)

Output:

File "<ipython-input-228-d1c2b3481864>", line 5

avg(name = 'Ali', 5, 9, 20, 34, 87, 112)

^

SyntaxError: positional argument follows keyword argument

Because the grammar of the language (Python) specifies that positional arguments appear before keyword arguments in calls.

**26. Write the output of each line, writedown the link between parameters and arguments. Remove one or two \*\* from other\_info and observe the ouput.**

**def display\_result(winner, score, \*\*other\_info):**

**print("The winner was " + winner + ".")**

**print("The score was " + score + ".")**

**display\_result(winner = "Manchester", score = "1-0", overtime = "yes", injuries= "none")**

Output:

The winner was Manchester.

The score was 1-0.

If we remove one or two \* we get :

TypeError: display\_result() got an unexpected keyword argument 'overtime'

As explained in detail above, in the last answer, because the \* before the parameter name allows arbitrary number of arguments to pass into the function.

To mark parameters as keyword-only, indicating the parameters must be passed by keyword argument, we have to place an \* in the arguments list. Removing both \*\* makes it so that **display\_result** takes 3 arguments but 4 were given.

When we remove both the asterisks it gives the error at the first keyword argument because then the parameter **other\_info** cannot accept arbitrary arguments and has to be specified by one key-word argument or a positional argument (parameter) properly positioned, i.e., before the keyword arguments (parameters):

def display\_result(winner, score, other\_info):

print("The winner was " + winner)

print("The score was " + score)

display\_result(winner = "Manchester", score = "1-0", other\_info = "yes")

Output:

The winner was Manchester

The score was 1-0

OR

def display\_result(other\_info, winner, score):

print("The winner was " + winner)

print("The score was " + score)

display\_result("yes", winner = "Manchester", score = "1-0", )

Output:

The winner was Manchester

The score was 1-0

\The \*\* operator does something similar, but with keyword arguments.\

If we remove one asterisk then *winner* and *score* become keyword-only parameters while other-info is a positional parameter and since positional arguments appear before keyword arguments in calls we get a syntax error but after properly positioning the arguments:

def display\_result(\*other\_info, winner, score):

print("The winner was " + winner)

print("The score was " + score)

display\_result(overtime = "yes", injuries= "none", winner = "Manchester", score = "1-0")

Output:

TypeError: display\_result() got an unexpected keyword argument 'overtime'

Because **other\_info** is a positional parameter it cannot accept keyword arguments so:

def display\_result(\*other\_info, winner, score):

print("The winner was " + winner)

print("The score was " + score)

display\_result("yes", "none", winner = "Manchester", score = "1-0")

Output:

The winner was Manchester

The score was 1-0

**27. The position of parameters and arguments is re-arranged. Just think and find the logic behind output or the error.**

**def display\_result(winner, \*\*other\_info, score):**

**print("The winner was " + winner)**

**print("The score was " + score)**

**display\_result(winner=“Manchester", overtime ="yes", injuries="none“ , score="1-0“ )**

Output:

SyntaxError: invalid syntax

Because positional arguments precede keyword arguments and \*\* unpacking. Explained in detail in the previous answer.

Ref: <https://www.python.org/dev/peps/pep-0448/>

So it must be written as in **26.**:

def display\_result(winner, score, \*\*other\_info):

print("The winner was " + winner + ".")

print("The score was " + score + ".")

display\_result(winner = "Manchester", score = "1-0", overtime = "yes", injuries= "none")

to get the correct

**Output:**

The winner was Manchester

The score was 1-0

**28. X = 123**

**print('%06d'%X)**

**Output:**

b) 000123

**29. X = 22.19**

**print('%5.2f'%X)**

**Output:**

c) 22.19

**30. '{0:f}, {1:2f}, {2:05.2f}'.format(1.23456, 1.23456, 1.23456)**

**Output:**

d) ‘1.234560, 1.234560, 01.23’

**f1** means first iteration, and so on

**31. Write down the output of each line after each iterations. Do multiple experiments to change values.**

# while loops use only Boolean expression and when it is true. So when it gets true it'll execute until it gets false. while (false) means the condition is false which will end the loop. while (True) means the condition is True which will continue the loop

**i = 1**

**while False:**

**if i%2 == 0:** # f1: 1 % 2 != 0 f2: 2 % 2 == 0

**break** # f1: 1 is not divisble by 2 so it doesn't break f2: 2 is divisble by 2, the condition

becomes true so the while False loop breaks

**print(i)** # i = 1

**i += 2** # i = i + 2 = 1+1 = 2 but it won't print until the next iteration of the loop, where it becomes divisible by 2 and the loop breaks because the while False loop only runs while the condition is false.

**Output:**

1

**32. Write down the output of each line after each iterations. Do multiple experiments to change values.**

**x = "abcdef"**

**i = "a"**

**while i in x:** # while 'a' is in 'abcdef'

**x = x[:-1]**  # run string x from index 0 to -1 (or 0-5), i.e., from a-f and put them into x

**print(i, end = " ")** # print i = 'a' from first to last index ending each iteration with an empty space character

**Output:**

a a a a a a

# prints i = 'a', a that is, from index 0 to -1 (0-5), i.e., six times because the while loop iterates six times from index 0 to -1 while i = 'a' is in x = 'abcdef'.

f1 : x = x[0:-1] = abcde

f2: x = x[0: -2] = abcd

f3: x = x[0: -3] = abc

f4: x = x[0: -4] = ab

f5: x = x[0: -5] = a

f6: x = x[0: -6] = ' '

----------------

**x = "abcdef"**

**i = "a"**

**while i in x:**

**x = x[:-1]**

**print(i, end = " ")**

**Output:**

a # just prints i = 'a' one time because outside the while loop

rest same as above

--------------

**x = "abcdef"**

**i = "a"**

**while i in x:**

**x = x[:-1]**

**print(x, end = " ")** # print x = 'abcdef' line by line from first to last index ending each iteration with an empty space character

Prints loop line by line because inside the while loop

**Output:**

abcde abcd abc ab a

f1 : x = x[0:-1] = abcde

f2: x = x[0: -2] = abcd

f3: x = x[0: -3] = abc

f4: x = x[0: -4] = ab

f5: x = x[0: -5] = a

f6: x = x[0: -6] = ' '

-------------

**x = "abcdef"**

**i = "a"**

**while i in x:**

**x = x[:-1]**

**print(x, end = " ")** # doesn't print anything because there is only an empty space from the last iteration ending with another empty space from the end argument of print

**x** # gives the empty space ' '

**Output:**

' '

# because the last iteration gives

f6: x = x[0: -6] = ' '

**33. Write down the output of each line after each iterations. Do multiple experiments to change values.**

**for i in ''.join(reversed(list('abcd'))):** # list('abcd') constructs a list and returns ['a', 'b', 'c', 'd']

whose items are the same and in the same order as iterable’s items. iterable may be either a sequence, a container that supports iteration, or an iterator object.

reverses the items of i in place

**print (i)**

**Output:**

d

c

b

a

Prints output in each new line because the print function by default ends with a new line, in reverse from 0th to last (3rd)index.