**Final Test**

Scroll to begin 

Well done! You've reached the end of the course and completed a major milestone in your Python programming education.

You are now prepared to take the ultimate challenge, the **Final Test**, which will help you review the most important information you've read, and test the skills and knowledge you've gained throughout the course.

The **Final Test** is based on what you have learned in the entire course. There are 35 questions in total, and you need to score at least 70% to pass. Good luck!

Complete Question 1

Question 1

Multiple choice question

What is the output of the following snippet?

my\_list = [1, 2]

for v in range(2):

    my\_list.insert(-1, my\_list[v])

print(my\_list)



[1, 2, 2, 2]



[1, 2, 1, 2]



[1, 1, 1, 2]



[2, 1, 1, 2]

Let’s analyze the program:

* Line 1: the *my\_list* list is created. The list stores the following contents: *[1, 2]*
* Line 3: the *range()* function generates a sequence of two numbers (*0* and *1*), and the *for* loop iterates over that sequence. The *v* variable takes the value of the number inside the sequence on each iteration, and the loop continues until the last number in the sequence is reached. The body of the loop is entered and the my\_list.insert(-1, my\_list[v]) operation is performed two times.
* Line 4: after the first *for* loop iteration (when *v = 0*), the *[1, 2]* list is updated – the *insert()* method inserts a new element to the list (*v = 0*, so *my\_list[v] = 1*) at the specified index (-1). The *my\_list* list after the first *for* loop iteration has the following contents: *[1, 1, 2]*. After the second for loop iteration (when *v = 1*), the *[1, 2, 3]* list is updated for the second time – the *insert()* method inserts another new element to the list (*v = 1*, so *my\_list[v] = 1*) at the specified index (*-1*). The *my\_list* list after the second *for* loop iteration contains the following contents: *[1, 1, 1, 2]*.
* Line 6: the *print()* function prints the list contents to the screen.

Complete Question 2

Question 2

Multiple choice question

The meaning of a *positional argument* is determined by:



the argument’s name specified along with its value



its position within the argument list



its value



its connection with existing variables

Positional arguments can be called by their position in a function call, and therefore must be included in the correct order.

Complete Question 3

Question 3

Multiple choice question

Which of the following sentences are **true** about the code? (Select **two** answers)

nums = [1, 2, 3]

vals = nums



nums and vals are different lists



nums has the same length as vals



vals is longer than nums



nums and vals are different names of the same list

Assigning *nums* to *vals* creates a situation in which the same list (i.e. *[1, 2, 3]*) has two different names. Giving a new name to an existing list, in our case *vals* to nums, is called **aliasing**. And since *nums* and *vals* are two different names that refer to the same object, they also have the same length (*3*).

Complete Question 4

Question 4

Multiple choice question

An operator able to check whether two values are **not equal** is coded as:



!=



=/=



not ==



<>

The *!=* operator checks whether two values are not equal. For example, the following expression *1 != 2* will evaluate as *True*.

Complete Question 5

Question 5

Multiple choice question

The following snippet:

def function\_1(a):

    return None

def function\_2(a):

    return function\_1(a) \* function\_1(a)

print(function\_2(2))



will output 16



will output 4



will output 2



will cause a runtime error

The *None* object has no arithmetic operators defined – the program will raise the *TypeError* exception, because the *NoneType* type cannot be the *\** operator’s operand.

Complete Question 6

Question 6

Multiple choice question

The result of the following division:

1 // 2



cannot be predicted



is equal to 0.5



is equal to 0



is equal to 0.0

The *//* operator divides two numbers and rounds the result down to the nearest integer number. Since *1* divided by *2* is 0.5, the resulting number is rounded down to *0*.

Complete Question 7

Question 7

Multiple choice question

The following snippet:

def func(a, b):

    return b \*\* a

print(func(b=2, 2))



is erroneous



will output 4



will output 2



will output None

The program will raise the *SyntaxError*, because the positional argument in the function call (*2*) follows the keyword argument (*b=2*). Changing the order of the arguments in the *func()* function invocation (i.e. *func(2, b=2)*) fixes the issue, in which case the program would output *4* as the result.

Complete Question 8

Question 8

Multiple choice question

What value will be assigned to the x variable?

z = 0

y = 10

x = y < z and z > y or y < z and z < y



False



1



True



0

Let’s analyze the example:

* *0* is assigned to *z*, and *10* is assigned to *y*,
* the result of the following evaluation y < z and z > y or y < z and z < y is assigned to *x*:
* *y < z* evaluates to False,
* *z > y* evaluates to False,
* *y < z* evaluates to False again,
* and *z < y* evaluates to True.
* And now: False **and** False is False; False **or** False is False; and finally False **and** True is False.

Complete Question 9

Question 9

Multiple choice question

Which of the following variable names are **illegal** and will cause the SyntaxError exception? (Select **two** answers)



print



for



In



in

The *in* and *for* names are Python reserved words (keywords). These cannot be used as variable names. Note that the name *print* is not a Python reserved word and can be used as a variable name, in which case it will shadow the Python built-in function, *print()*.

Complete Question 10

Question 10

Multiple choice question

What is the output of the following snippet?

my\_list = [x \* x for x in range(5)]

def fun(lst):

    del lst[lst[2]]

    return lst

print(fun(my\_list))



[0, 1, 4, 9]



[0, 1, 9, 16]



[1, 4, 9, 16]



[0, 1, 4, 16]

Let’s analyze the example:

* Line 1: The list comprehension is used to create the following list: *[0, 1, 4, 9, 16]*, and assign it to the *my\_list* variable,
* Lines 4-6: The *fun()* function is defined, which takes a list (*lst*) as its argument. In the body of the function, the *del* instruction deletes a specified element of the list, and returns the updated list,
* Line 9: The *print()* function takes the *fun()* function as its argument, and prints the returned value to the screen. The *fun()* function takes the *my\_list* list as its argument and processes it – the *del* instruction deletes the fourth element of the list (*my\_list[2]* is *4*).

Complete Question 11

Question 11

Multiple choice question

What is the output of the following piece of code?

x = 1

y = 2

x, y, z = x, x, y

z, y, z = x, y, z

print(x, y, z)



1 1 2



2 1 2



1 2 2



1 2 1

Let’s analyze the example:

* *x = 1* and *y = 2*
* as a result of the first multiple variable assignment (Line 3), ***x = 1***, *y = 1*, and *z = 2*
* as a result of the second multiple variable assignment (Line 4), *z = x = 1*, ***y = y = 1***, and ***z = z = 2*** (because the original assignment from the previous line now overwrites the first *z* assignment from the current line)
* the result printed to the screen is therefore 1 1 2

Complete Question 12

Question 12

Multiple choice question

What will be the output of the following snippet?

a = 1

b = 0

a = a ^ b

b = a ^ b

a = a ^ b

print(a, b)



1 0



1 1



0 0



0 1

Let’s analyze the code line by line:

* Line 1: *1* is assigned to variable *a*
* Line 2: *0* is assigned to variable *b*
* Line 3: the result of the evaluation is *1*, so *a = 1*
* Line 4: the result of the evaluation is also *1*, so *b = 1*
* Line 5: the result of the evaluation *a = a ^ b* (where *a = 1* and *b = 1*) is *0*, so *a = 0*
* the new values assigned to the *a* and *b* variables are then printed to the console.

Complete Question 13

Question 13

Multiple choice question

What is the output of the following snippet?

def fun(x):

    if x % 2 == 0:

        return 1

    else:

        return 2

print(fun(fun(2)))



2None



1



2



the code will cause a runtime error

The *print()* function in line 8 takes the *fun()* function as its argument, which takes another *fun()* function as its argument, which in turn takes the value *2* as its argument.

Let’s analyze the function calls, starting with the right-most ones:

* the *fun(2)* call – *x* is assigned the value of *2*, so *x % 2 = 0*, so the comparison *0 == 0* evaluates to True, and the function returns *1*, which is passed to the “left” *fun()* function as its argument,
* the *fun(1)* call – *x* is assigned the value of *1*, so *x % 2 = 1*, so the comparison *1 == 0* evaluates to False, which means the function returns *2*, which is passed to the *print()* function as its argument. Therefore, *2* is printed to the screen.

Complete Question 14

Question 14

Multiple choice question

Take a look at the snippet and choose the **true** statement:

nums = [1, 2, 3]

vals = nums

del vals[:]



nums is longer than vals



vals is longer than nums



the snippet will cause a runtime error



nums and vals have the same length

Assigning *nums* to *vals* creates a situation in which the same list (i.e. *[1, 2, 3]*) has two different names. Giving a new name to an existing list, in our case *vals* to nums, is called **aliasing**. And since *nums* and *vals* are two different names that refer to the same object, they are also the same length.

The *del* instruction empties the list pointed to by *nums* and *vals*, which means the list has a zero length, which in turn means that *nums* and *vals* are the same length.

Complete Question 15

Question 15

Multiple choice question

What is the output of the following piece of code if the user enters two lines containing 3 and 2 respectively?

x = int(input())

y = int(input())

x = x % y

x = x % y

y = y % x

print(y)



1



3



0



2

Let’s analyze what happens:

* *x = 3* and *y = 2*
* Line 3: *x = 3 % 2 = 1*
* Line 4: *x = 1 % 2 = 1*
* Line 5: *x = 2 % 1 =* ***0***
* the result is printed to the screen.

Complete Question 16

Question 16

Multiple choice question

What is the output of the following piece of code if the user enters two lines containing 3 and 6 respectively?

y = input()

x = input()

print(x + y)



6



3



36



63

It’s a tricky question because of two reasons:

* Reason one: the *input()* function converts the data inputted by the user to a string, and the result of adding two strings to each other is gluing them together: *"string" + "string" = "stringstring"* (concatenation).
* Reason two: the first user input is assigned to the *y* variable, while the second to the *x*variable. However, they are printed in the reverse order.

Complete Question 17

Question 17

Multiple choice question

What is the output of the following piece of code?

print("a", "b", "c", sep="sep")



asepbsepcsep



abc



a b c



asepbsepc

The *print()* function prints the strings *"a"*, *"b"*, and *"c"* to the screen, and separates them with the "sep" string. The *sep* keyword parameter determines the type of separator used between the subsequent *print()* arguments outputted to the screen.

Complete Question 18

Question 18

Multiple choice question

What is the output of the following piece of code?

x = 1 // 5 + 1 / 5

print(x)



0.5



0.4



0.2



0

Remember the principle of Operator Precedence, and the difference between the two Python division operators: // (integer division) and */* (floating-point division). Let’s analyze the expression in the first line:

Because the division operators have a higher priority than the addition operator, we can add brackets for readability, and evaluate the expression in the following way: *(1 // 5) + (1 / 5)* gives *0 + 0.2*, which in turn evaluates to *0.2*.

Complete Question 19

Question 19

Multiple choice question

Assuming that my\_tuple is a correctly created tuple, the fact that tuples are immutable means that the following instruction:

my\_tuple[1] = my\_tuple[1] + my\_tuple[0]



is fully correct



may be illegal if the tuple contains strings



can be executed if and only if the tuple contains at least two elements



is illegal

Tuples are immutable sequences, which means you cannot update them directly – adding, changing, or removing tuple items requires different types of operations to be performed, for example converting the tuple to a list, updating the list, and converting it back to a tuple.

Complete Question 20

Question 20

Multiple choice question

What is the output of the following piece of code if the user enters two lines containing 2 and 4 respectively?

x = float(input())

y = float(input())

print(y \*\* (1 / x))



4.2



0.0



2.0



1.0

Let’s analyze what happens:

* the user enters *2*, which is converted to a float, and assigned to the *x* variable
* the user enters *4*, which is converted to a float, and assigned to the *y* variable
* the result of the following evaluation is printed to the screen: *4 \*\* (1 / 2) = 4 \*\* 0.5 = 2.0*

Complete Question 21

Question 21

Multiple choice question

What is the output of the following snippet?

dct = {'one': 'two', 'three': 'one', 'two': 'three'}

v = dct['three']

for k in range(len(dct)):

    v = dct[v]

print(v)



two



one



('one', 'two', 'three')



three

Let’s analyze the code:

* Line 1: the *dct* dictionary containing three key:value pairs is created
* Line 2: the *v* variable is created and assigned the dictionary value associated with the *'three'* key, that is the string *"one"*
* Lines 4-5: the *for* loop performs three iterations (the length of the *dct* ctionary is *3*):
* after *k = 0* *v = "two"*
* after *k = 1* *v = "three"*
* after *k = 2* *v = "one"*
* the string "one" assigned to the *v* variable is then printed to the screen.

Complete Question 22

Question 22

Multiple choice question

How many elements does the lst list contain?

lst = [i for i in range(-1, -2)]



three



zero



two



one

The *lst* list is empty, because the *range()* function, which consists in of the list comprehension, sets the *start* parameter with the initial value of *-1* and the *stop* parameter with the initial value of *-2* (the *stop* parameter has a lower value than the *start* parameter, which makes it impossible to create a sequence).

Complete Question 23

Question 23

Multiple choice question

Which of the following lines **correctly** invoke the function defined below? (Select **two** answers)

def fun(a, b, c=0):

    # Body of the function.



fun()



fun(b=1)



fun(b=0, a=0)



fun(0, 1, 2)

Because the *c* argument is a keyword argument with a specified default value, it’s enough to call the function by specifying default values for the remaining two keyword arguments, *a* and *b*. It’s also possible to replace a keyword parameter default value with a function call with positional arguments: *fun(0, 1, 2)*. In both cases, it’s important that all the function parameters be served.

The *fun(b=1)* call is incorrect because it does not specify a value for the *a* argument required in the function definition. The *fun()* call is incorrect because it does not specify the values for the *a* and *b* arguments required in the function definition.

Complete Question 24

Question 24

Multiple choice question

What is the output of the following snippet?

def fun(x, y):

    if x == y:

        return x

    else:

        return fun(x, y-1)

print(fun(0, 3))



the snippet will cause a runtime error



2



0



1

The following snippet is an example of a recursive function.

The following two arguments are sent to the *fun(x, y)* function: *0* and *3*. The result of the comparison *x == y* is False, so the *else:* block is triggered.

In the *else:* block, the *fun(x, y-1)* function call is returned, and this time the following two arguments are sent: *x = 0*, and *y = 3 - 1 =* ***2***. The process is repeated (*x = 0*, *y = 2 – 1 = 1*, and *x = 0*, *y = 1 – 1 = 0) until the value assigned to the y variable is equal to 0, in which case the result of the x == ycomparison is True, and the function returns the x value of 0.*

Complete Question 25

Question 25

Multiple choice question

How many stars (\*) will the following snippet send to the console?

i = 0

while i < i + 2 :

    i += 1

    print("\*")

else:

    print("\*")



one



two



the snippet will enter an infinite loop, printing one star per line



zero

Because *i = 0*, the result of the following comparison: *i < 1 +2* is True, so the program enters the *while* loop. However, in the body of the loop *i* is then incremented, not decremented, which means the result of the comparison *i < 1 +2* will always evaluate to True, which means the loop will perform an infinite number of iterations, executing the *print("\*")* function with each and every iteration.

Complete Question 26

Question 26

Multiple choice question

What is the output of the following snippet?

tup = (1, 2, 4, 8)

tup = tup[-2:-1]

tup = tup[-1]

print(tup)



4



(4)



44



(4,)

Let’s analyze the code line by line:

* Line 1: the four-element *tup* tuple is created
* Line 2: tuples can be sliced, so the following slice result is assigned to *tup*: *(4,)*
* Line 3: since the *tup* tuple is a one-element tuple now *(4,)*, indexing the last and only element will “extract” it from the tuple, and assign to the *tup* variable as an integer value.
* the value *4* will then be printed to the screen.

Complete Question 27

Question 27

Multiple choice question

What is the output of the following snippet?

dd = {"1": "0", "0": "1"}

for x in dd.vals():

    print(x, end="")



0 1



0 0



1 0



the code is erroneous (the dict object has no vals() method)

The following snippet will raise the *AttributeError* exception, because the *dict* object has no attribute named *vals*. Replacing *vals* with *values* should do the job.

Complete Question 28

Question 28

Multiple choice question

What is the output of the following snippet?

dct = {}

dct['1'] = (1, 2)

dct['2'] = (2, 1)

for x in dct.keys():

    print(dct[x][1], end="")



21



(2,1)



12



(1,2)

Let’s analyze the code:

* Line 1: an empty dictionary is created
* Line 2-3: the key:value pairs are created, and added to the *dct* dictionary:  
  *dct = {'1': (1, 2), '2': (2, 1)}*
* Line 5-6: perform two *for* loop iterations (iterate through the *dct* keys) and print the dictionary values (tuple elements) indexed in the *print()* function: *x = 1* *2*, *x = 2* *1*
* the *end=""* parameter joins the two printed lines together.

Complete Question 29

Question 29

Multiple choice question

What is the output of the following snippet?

def fun(inp=2, out=3):

    return inp \* out

print(fun(out=2))



the snippet is erroneous and will cause SyntaxError



6



4



2

The default value (*3*) assigned to the *out* argument of the *fun()* function definition has been changed to *2* in the *fun()* function call passed as the argument to the *print()* function. This way the result of the evaluation *(inp = 2) \* (out = 2)* is equal to 4.

Complete Question 30

Question 30

Multiple choice question

How many hashes (#) will the following snippet send to the console?

lst = [[x for x in range(3)] for y in range(3)]

for r in range(3):

    for c in range(3):

        if lst[r][c] % 2 != 0:

            print("#")



zero



nine



three



six

Let’s analyze the code:

* Line 1: we’re using a list comprehension to create the following two-dimensional array:  
  [[0, 1, 2], [0, 1, 2], [0, 1, 2]]
* the program enters the first *for* loop with the *r* iteration variable taking the value of the first item inside the sequence generated by the *range(3)* function, which is *0*. Then the program enters the second (nested) *for* loop with the *c* iteration variable taking the value of the first item inside the sequence generated by the *range(3)* function, which is *0*. The following condition is checked:  
  if lst[0][0] % 2 != 0, then print a hash (#) to the screen. The value returned from the *lst* list stored under the *[0][0]* indexes is *0*, so the condition evaluates to false (*0 % 2 != 0* → *False*), which means a hash is not printed to the screen at this time, and the program execution jumps back to the outer loop. The iterations are repeated for the following index pairs: *[0][1]*, *[0][2]*, *[1][2]*, *[1][0]*, *[1][1]*, *[1][2]*, *[2][2]*, *[2][0]*, *[2][1]*, and *[2][2]*.
* the *print()* function will print a hash when the following index pairs become part of the conditional check:
* *[0][1]* → *1* because *if 1 % 2 != 0* evaluates to True
* *[1][1]* → *1* because *if 1 % 2 != 0* evaluates to True
* *[2][1]* → *1* because *if 1 % 2 != 0* evaluates to True
* Therefore, the *print()* function will output three hashes to the screen.

Complete Question 31

Question 31

Multiple choice question

What is the output of the following code if the user enters a 0?

try:

    value = input("Enter a value: ")

    print(int(value)/len(value))

except ValueError:

    print("Bad input...")

except ZeroDivisionError:

    print("Very bad input...")

except TypeError:

    print("Very very bad input...")

except:

    print("Booo!")



0.0



1.0



Very very bad input...



Very bad input...



Booo!



Bad input...

The program will print *0.0* because the following expression *int(0) / len("0")* evaluates to *0.0* (*0 / 1* → *0.0*), so the program flow does not enter any of the *except* branches, and no exception is raised.

Complete Question 32

Question 32

Multiple choice question

What is the expected behavior of the following program?

try:

    print(5/0)

    break

except:

    print("Sorry, something went wrong...")

except (ValueError, ZeroDivisionError):

    print("Too bad...")



The program will cause a SyntaxError exception.



The program will cause a ZeroDivisionError exception and output a default error message.



The program will raise an exception handled by the first except block.



The program will cause a ValueError exception and output a default error message.



The program will cause a ZeroDivisionError exception and output the following message: Too bad...



The program will cause a ValueError exception and output the following message: Too bad...

The program will raise a *SyntaxError* exception, because the *break* statement used in the *try* block violates Python’s grammar rules (the *break* instruction must be used in a loop). When the code is run, before it is executed, the Python interpreter first parses it and converts it to Python byte code. If the parser encounters a syntax error which has a fatal effect on the program, it won’t be able to parse the code successfully, and will raise a *SyntaxError*.

Remember that the parser analyzes the program before its execution, so if it encounters a syntax error, the program will not be executed. If your code has no syntax errors, the program is executed and may raise other types of exceptions.

Complete Question 33

Question 33

Multiple choice question

What is the expected behavior of the following program?

foo = (1, 2, 3)

foo.index(0)



The program will cause a ValueError exception.



The program will cause an AttributeError exception.



The program will cause a TypeError exception.



The program will output 1 to the screen.



The program will cause a SyntaxError exception.

The program will raise a *ValueError* exception, because the tuple *index()* method returns the index of a given element in a tuple, and the *foo* tuple does not have the *0* element.

Full error message: *ValueError: tuple.index(x): x not in tuple*.

Complete Question 34

Question 34

Multiple choice question

Which of the following snippets shows the correct way of handling multiple exceptions in a single *except* clause?

# A:

except (TypeError, ValueError, ZeroDivisionError):

    # Some code.

# B:

except TypeError, ValueError, ZeroDivisionError:

    # Some code.

# C:

except: (TypeError, ValueError, ZeroDivisionError)

    # Some code.

# D:

except: TypeError, ValueError, ZeroDivisionError

    # Some code.

# E:

except (TypeError, ValueError, ZeroDivisionError)

    # Some code.

# F:

except TypeError, ValueError, ZeroDivisionError

    # Some code.



A, C, and D



B and C



D and E



F only



A and F



A only



A and B

The following syntax:  
  
except (TypeError, ValueError, ZeroDivisionError):  
  
is the only correct option for handling multiple built-in exceptions within a single *except* clause.

Complete Question 35

Question 35

Multiple choice question

What will happen when you attempt to run the following code?

print(Hello, World!)



The code will raise the *ValueError* exception.



The code will raise the *SyntaxError* exception.



The code will raise the *TypeError* exception.



The code will raise the *AttributeError* exception.



The code will print Hello, World! to the console.

The code will raise a *SyntaxError* exception, because the string argument passed to the *print()* function must be delimited by quotes.

The program does not raise a *NameError* exception, because the name *World!* contains an illegal character: !, and *Hello* and *World!* cannot be considered variable names that have not been recognized by Python.

You've submitted your answers!

77%

Assessment results.

You've scored 77%.

Congratulations, you have passed the assessment.