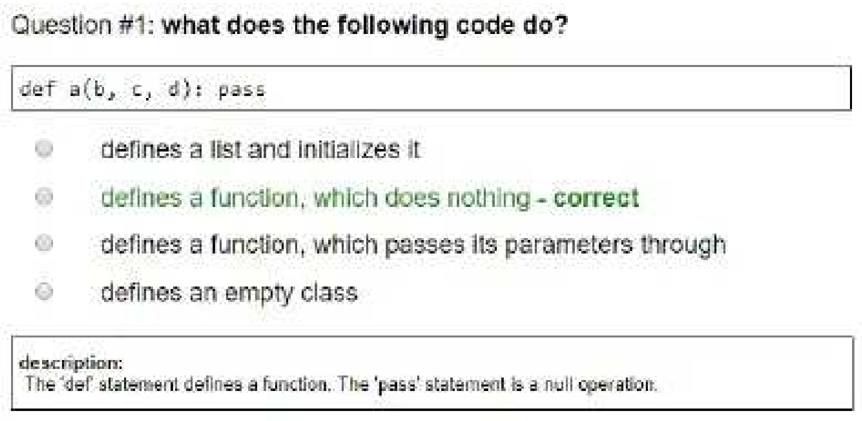
GRILE Python

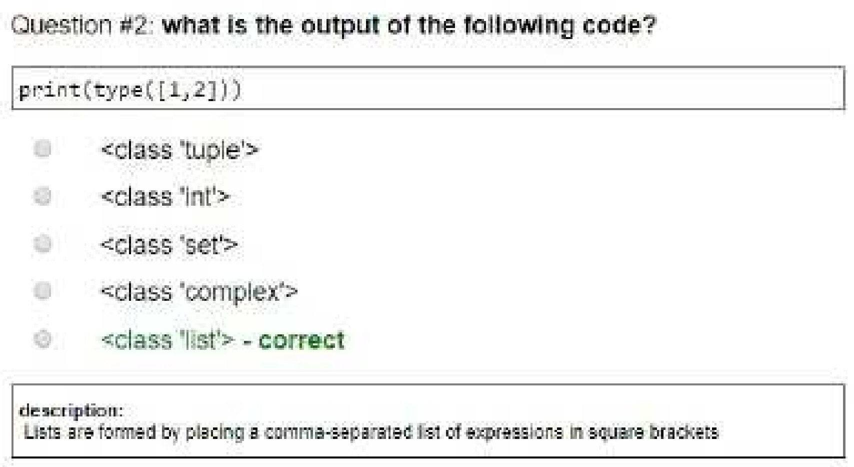


Question #1: **what does the following code do?**

```
def a(b, c, d): pass
```

- defines a list and initializes it
- defines a function, which does nothing - **correct**
- defines a function, which passes its parameters through
- defines an empty class

description:
The 'def' statement defines a function. The 'pass' statement is a null operation.

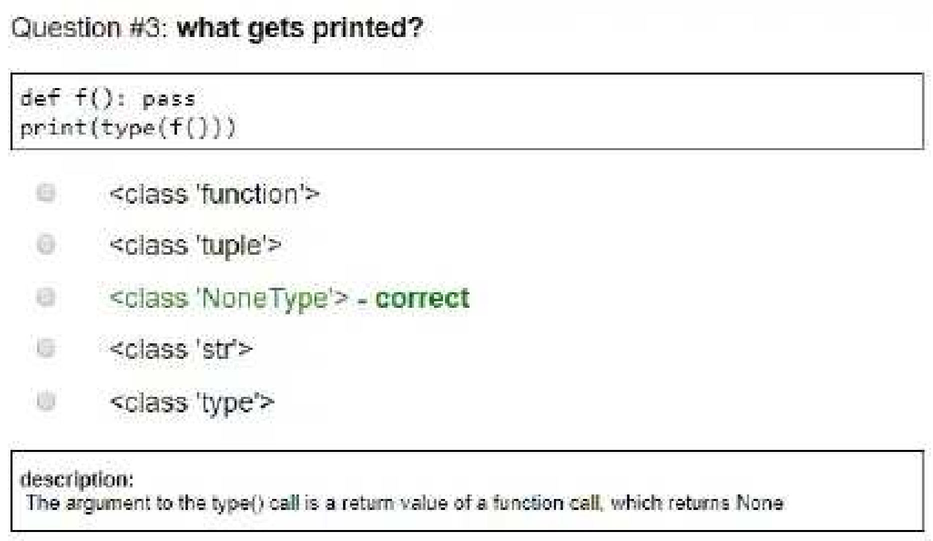


Question #2: **what is the output of the following code?**

```
print(type([1,2]))
```

- <class 'tuple'>
- <class 'int'>
- <class 'set'>
- <class 'complex'>
- <class 'list'> - **correct**

description:
Lists are formed by placing a comma-separated list of expressions in square brackets

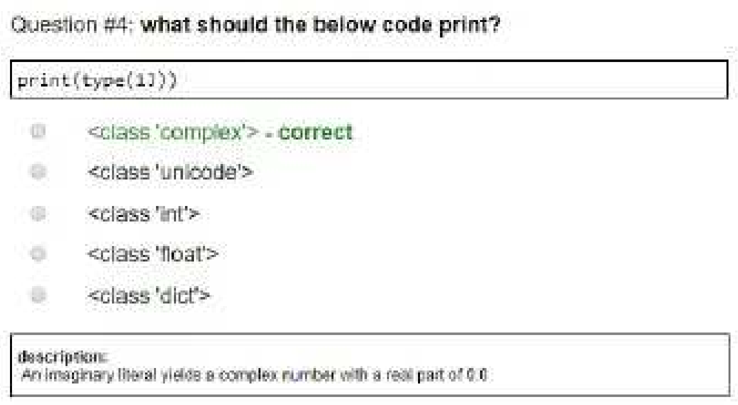


Question #3: **what gets printed?**

```
def f(): pass
print(type(f()))
```

- <class 'function'>
- <class 'tuple'>
- <class 'NoneType'> - **correct**
- <class 'str'>
- <class 'type'>

description:
The argument to the type() call is a return value of a function call, which returns None



Question #4: **what should the below code print?**

```
print(type(1J))
```

- <class 'complex'> - **correct**
- <class 'unicode'>
- <class 'int'>
- <class 'float'>
- <class 'dict'>

description:
An imaginary literal yields a complex number with a real part of 0.0

Question #5: **what is the output of the following code?**

```
print(type(lambda:None))
```

- ○ <class 'NoneType'>
- ○ <class 'tuple'>
- ○ <class 'type'>
- ○ <class 'function'> - **correct**
- ○ <class 'bool'>

description:
'lambda arguments: expression' yields a function object

Question #6: **what is the output of the below program?**

```
a = [1,2,3,None,(),[],]
print(len(a))
```

- ○ syntax error
- ○ 4
- ○ 5
- ○ 6 - **correct**
- ○ 7

description:
The trailing comma in the list is ignored, the rest are legitimate values

Question #7: **what gets printed?**

```
print(type(1/2))
```

- ○ <class 'int'>
- ○ <class 'number'>
- ○ <class 'float'> - **correct**
- ○ <class 'double'>
- ○ <class 'tuple'>

description:
division of an integer by another integer yields a float

Question #8: **What gets printed?**

```
d = lambda p: p * 2
t = lambda p: p * 3
x = 2
x = d(x)
x = t(x)
x = d(x)
print(x)
```

- ○ 7
- ○ 12
- ○ 24 - **correct**
- ○ 36
- ○ 48

description:
start with 2, multiply by 2, multiply by 3, multiply by 2.

Question #9: **What gets printed?**

```
x = 4.5
y = 2
print(x//y)
```

- ○ 2.0 - **correct**
- ○ 2.25
- ○ 9.0
- ○ 20.25
- ○ 21

description:
this is truncating division. The remainder is dropped

Question #10: **What gets printed?**

```
nums = set([1,1,2,3,3,3,4])
print(len(nums))
```

- ○ 1
- ○ 2
- ○ 4 - **correct**
- ○ 5
- ○ 7

description:
nums is a set, so only unique values are retained.

Question #11: **What gets printed?**

```
x = True
y = False
z = False

if x or y and z:
    print("yes")
else:
    print("no")
```

- ○ yes - **correct**
- ○ no
- ○ fails to compile

description:
AND is higher precedence than OR in python and is evaluated first

Question #12: **What gets printed?**

```
x = True
y = False
z = False

if not x or y:
    print(1)
elif not x or not y and z:
    print(2)
elif not x or y or not y and x:
    print(3)
else:
    print(4)
```

- ○ 1
- ○ 2
- ○ 3 - **correct**
- ○ 4

Question #14: **What gets printed?**

```
counter = 1

def doLotsOfStuff():

    global counter

    for i in (1, 2, 3):
        counter += 1

doLotsOfStuff()

print(counter)
```

- 1
- 3
- 4 - **correct**
- 7
- none of the above

Question #15: **What gets printed?**

```
print(r"\nwoow")
```

- new line then the string: woow
- the text exactly like this: r"\nwoow"
- the text like exactly like this: \nwoow - **correct**
- the letter r and then newline then the text: woow
- the letter r then the text like this: nwoow

description:
When prefixed with the letter 'r' or 'R' a string literal becomes a raw string and the escape sequences such as \n are not converted.

Question #19: **What gets printed?**

```
class Account:
    def __init__(self, id):
        self.id = id
        id = 666

acc = Account(123)
print(acc.id)
```

- None
- 123 - **correct**
- 666
- SyntaxError, this program will not run

Question #20: **What gets printed?**

```
name = "snow storm"

print(name[6:8])
```

- st
- sto
- to - **correct**
- tor
- Syntax Error

description:
This is a slice of a string from index 6 to index 8 not including index 8. The first character in the string is position 0.

Question #21: **What gets printed?**

```
name = "snow storm"

name[5] = 'X'

print(name)
```

- snow storm
- snowXstorm
- snow Xtorm
- ERROR, this code will not run - **correct**

Question #22: **Which numbers are printed?**

```
for i in range(2):
    print(i)

for i in range(4,6):
    print(i)
```

- 2, 4, 6
- 0, 1, 2, 4, 5, 6
- 0, 1, 4, 5 - **correct**
- 0, 1, 4, 5, 6, 7, 8, 9
- 1, 2, 4, 5, 6

Question #23: **What sequence of numbers is printed?**

```
values = [1, 2, 1, 3]
nums = set(values)

def checkit(num):
    if num in nums:
        return True
    else:
        return False

for i in  filter(checkit, values):
    print(i)
```

- 1 2 3
- 1 2 1 3 - **correct**
- 1 2 1 3 1 2 1 3
- 1 1 1 1 2 2 3 3
- Syntax Error

Question #24: **What sequence of numbers is printed?**

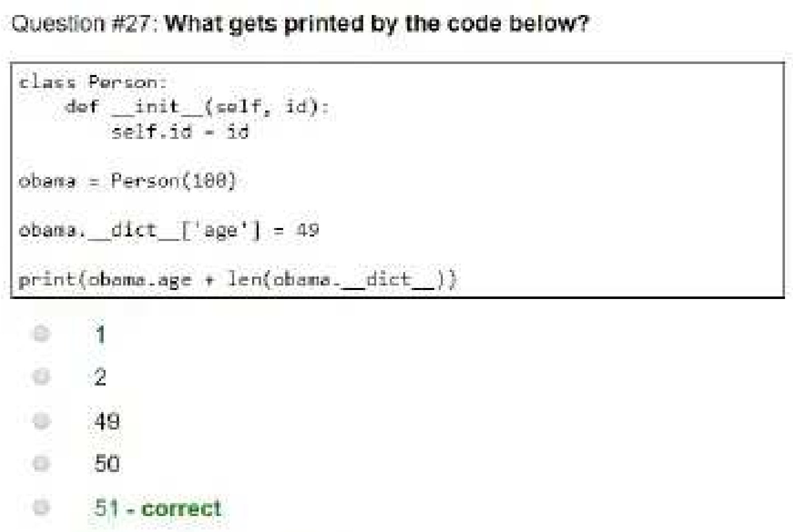
```
values = [2, 3, 2, 4]

def my_transformation(num):
    return num ** 2

for i in  map(my_transformation, values):
    print(i)
```

- 2 3 2 4
- 4 6 4 8
- 1 1.5 1 2
- 1 1 1 2
- 4 9 4 16 - **correct**

Question #26: **What gets printed by the code snippet below?**

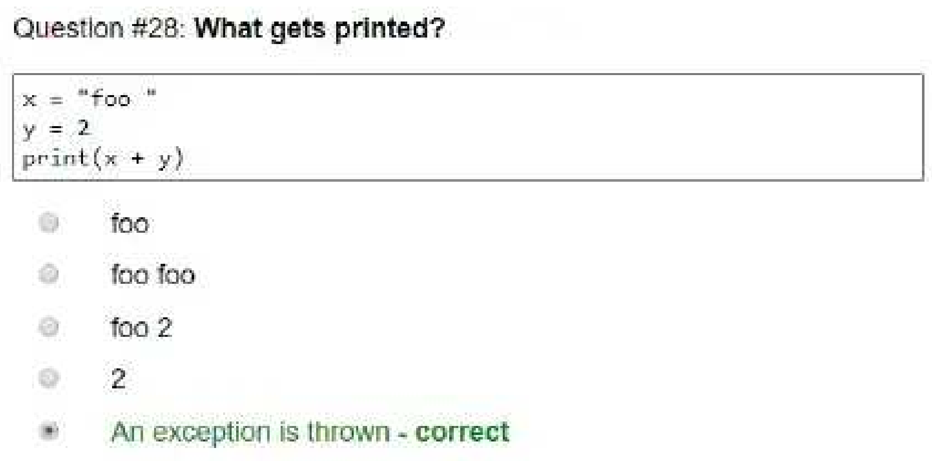
```
import math

print(math.floor(5.5))
```

- 0
- 5 - **correct**
- 5.5
- 6
- 6.0


Question #27: **What gets printed by the code below?**

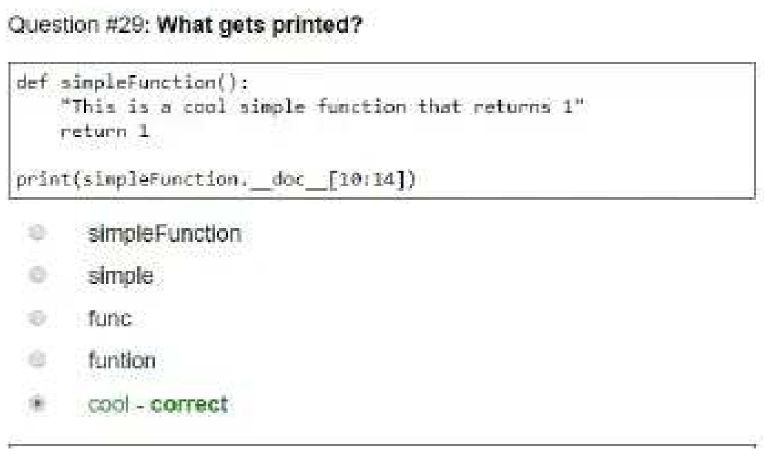
```
class Person:
    def __init__(self, id):
        self.id = id

obama = Person(100)

obama.__dict__['age'] = 49

print(obama.age + len(obama.__dict__))
```

- 1
- 2
- 49
- 50
- 51 - **correct**


Question #28: **What gets printed?**

```
x = "foo "
y = 2
print(x + y)
```

- foo
- foo foo
- foo 2
- 2
- [✓ An exception is thrown - **correct**]


Question #29: **What gets printed?**

```
def simpleFunction():
    "This is a cool simple function that returns 1"
    return 1

print(simpleFunction.__doc__[10:14])
```

- simpleFunction
- simple
- func
- funtion
- [✓ cool - **correct**]

Question #70: **What gets printed?**

```
kvps = { '1' : 1, '2' : 2 , '3' : 3, '4' : 4, '5' : 5}
newData = { '1' : 10, '3' : 30 }

kvps.update(newData)

x = sum(kvps.values())

print(x)
```

- 15
- 51 - **correct**
- 150
- An exception is thrown

description:
the update method of dictionary will update the values that have the same keys as the newData with newData's values.

Question #69: **What gets printed?**

```
kvps = { '1' : 1, '2' : 2 }
theCopy = dict(kvps)

kvps['1'] = 5

sum = kvps['1'] + theCopy['1']
print(sum)
```

- 1
- 2
- 6 - **correct**
- 10
- An exception is thrown

description:
Initializing one dictionary from another makes an independent copy, so changing one doesn't affect the other one

Question #68: **What gets printed?**

```
import copy

aList = [1,2]
bList = [3,4]

kvps = { '1' : aList, '2' : bList }
theCopy = copy.deepcopy(kvps)

kvps['1'][0] = 5

sum = kvps['1'][0] + theCopy['1'][0]
print(sum)
```

- 1
- 2
- 6 - **correct**
- 10
- An exception is thrown

description:
A deep copy will copy all the keys and values inside a dictionary. Therefore the list inside the dictionary are different in the first and second dictionaries of this example.

Question #67: **What gets printed**

```
aList = [1,2]
bList = [3,4]

kvps = { '1' : aList, '2' : bList }
theCopy = kvps.copy()

kvps['1'][0] = 5

sum = kvps['1'][0] + theCopy['1'][0]
print(sum)
```

- 1
- 2
- 6
- 10 - **correct**
- An exception is thrown

description:
The copy method provides a shallow copy therefore the list being held as the value inside the dictionary is the same list in the copy as the original.

Question #66: **What gets printed?**

```
kvps = { '1' : 1, '2' : 2 }
theCopy = kvps.copy()

kvps['1'] = 5

sum = kvps['1'] + theCopy['1']
print(sum)
```

- ○ 1
- ○ 2
- ○ 6 - correct
- ○ 10
- ○ An exception is thrown

description:
The copy method of the dictionary will make a new (shallow) copy of the dictionary so a change to the original in this case does not change the copy.

Question #65: **What gets printed?**

```
kvps = { '1' : 1, '2' : 2 }
theCopy = kvps

kvps['1'] = 5

sum = kvps['1'] + theCopy['1']
print(sum)
```

- ○ 1
- ○ 2
- ○ 7
- ○ 10 - correct
- ○ An exception is thrown

description:
Assignment will provide another reference to the same dictionary. Therefore the change to the original dictionary also is changing the copy.

Question #64: **What gets printed?**

```
x = sum(range(5))
print(x)
```

- 4
- 5
- [✓ 10 - **correct**]
- 15
- An exception is thrown

description:
range(5) produces a list of the numbers 0, 1, 2, 3, 4.

sum will add all the numbers in the list.

Question #62: **What gets printed?**

```
one = chr(104)
two = chr(105)
print(one + two)
```

- hi - **correct**
- h
- None
- 104105
- 209

description:
chr is a built in function that converts an ascii code to a 1 letter string.

Question #61: **What gets printed?**

```
class NumFactory:
    def __init__(self, n):
        self.val = n
    def timesTwo(self):
        self.val *= 2
    def plusTwo(self):
        self.val += 2

f = NumFactory(2)
for m in dir(f):
    mthd = getattr(f,m)
    if callable(mthd):
        mthd()

print(f.val)
```

- ○ 2
- ○ 4
- ○ 6
- ○ 8
- ○ An exception is thrown - **correct**

description:
An exception will be thrown when trying to call the __init__ method of the object without any parameters:

TypeError: __init__() takes exactly 2 arguments (1 given)

Question #60: **What gets printed?**

```
class A:
    def __init__(self, a, b, c):
        self.x = a + b + c

a = A(1,2,3)
b = getattr(a, 'x')
setattr(a, 'x', b+1)
print(a.x)
```

- ○ 1
- ○ 2
- ○ 3
- ○ 6
- ○ 7 - **correct**

description:
getattr can be used to get the value of a member variable of an object. setattr can be used to set it.

Question #59: **How do you create a package so that the following reference will work?**

```
p = mytools.myparser.MyParser()
```

- Declare the myparser package in mytools.py
- Create an __init__.py in the home dir
- Inside the mytools dir create a __init__.py and myparser.py - **correct**
- Create a myparser.py directory inside the mytools directory
- This can not be done

description:
In order to create a package create a directory for the package name and then put an __init__.py file in the directory.

Question #58: **What gets printed?**

```
def myfunc(x, y, z, a):
    print(x + y)

nums = [1, 2, 3, 4]

myfunc(*nums)
```

- 1
- 3 - **correct**
- 6
- 10
- An exception is thrown

description:
*nums will unpack the list into individual elements to be passed to the function.

Question #57: **What gets printed?**

```
def dostuff(param1, **param2):
   print(type(param2))


dostuff('capitals', Arizona='Phoenix',
California='Sacramento', Texas='Austin')
```

- in
- <class 'str'>
- <class 'tuple'>
- <class 'list'>
- <class 'dict'> - **correct**

description:
param2 aggregates the remaining parameters into a dictionary.

Question #56: **What gets printed?**

```
def dostuff(param1, *param2):
   print(type(param2))

dostuff('apples', 'bananas', 'cherry', 'dates')
```

- <class 'str'>
- <class 'int'>
- <class 'tuple'> - **correct**
- <class 'list'>
- <class 'dict'>

description:
param2 aggregates remaining parameters into a tuple.

Question #55: **What gets printed?**

```
def print_header(str):
    print("+++" + str + "+++")


print_header.category = 1
print_header.text = "some info"

print_header("{} and {}".format(print_header.category, print_header.text))
```

- ○ +++1 and some info+++ - **correct**
- ○ +++%s+++
- ○ +++1
- ○ 1
- ○ some info

description:
You can assign arbitrary typed information to functions.

Question #54: **What gets printed?**

```
a = 1
b = 2
a,b = b,a

output = "{} {}".format(a, b)
print(output)
```

- ○ 1 2
- ○ 2 1 - **correct**
- ○ An exception is thrown
- ○ This program has undefined behavior

description:
This is valid python code. This is assignment multiple variables at once. The values in b and a are being assigned to a and b, respectively.

Question #53: **What gets printed?**

```
my_tuple = (1, 2, 3, 4)
my_tuple.append( (5, 6, 7) )
print(len(my_tuple))
```

- ○ 1
- ○ 2
- ○ 5
- ○ 7
- ○ An exception is thrown - **correct**

description:
Tuples are immutable and don't have an append method. An exception is thrown in this case.

Question #52: **What gets printed?**

```
def addItem(listParam):
    listParam += [1]

mylist = [1, 2, 3, 4]
addItem(mylist)
print(len(mylist))
```

- ○ 1
- ○ 4
- ○ 5 - **correct**
- ○ 8
- ○ An exception is thrown

description:
The list is passed by reference to the function and modifications to the function parameter also affect the original list

Question #51: **What gets printed?**

```
list1 = [1, 2, 3, 4]
list2 = [5, 6, 7, 8]

print(len(list1 + list2))
```

- 2
- 4
- 5
- 8 - **correct**
- An exception is thrown

description:
The + operator appends the elements in each list into a new list.

Question #50: **Which of the following data structures can be used with the "in" operator to check if an item is in the data structure?**

- list
- set
- dictionary
- All of the above - **correct**
- None of the above

description:
The "in" operator can be used with all 3 of these data structures.

Question #49: **What gets printed?**

```
numbers = [1, 2, 3, 4]

numbers.append([5,6,7,8])

print(len(numbers))
```

- 4
- 5 - **correct**
- 8
- 12
- An exception is thrown

description:
When a list is passed to the append method of list, the entire list is added as an element of the list. The lists are not merged.

Question #48: **What gets printed?**

```
names1 = ['Amir', 'Barry', 'Chales', 'Dao']
names2 = [name.lower() for name in names1]

print(names2[2][0])
```

- i
- a
- c - **correct**
- C
- An exception is thrown

description:
List Comprehensions are a shorthand to creating a new list with the all the values in a original list modified by some python expression.

Question #47: **What gets printed?**

```
names1 = ['Amir', 'Barry', 'Chales', 'Dao']

if 'amir' in names1:
    print(1)
else:
    print(2)
```

- 1
- [✓ 2 - **correct**]
- An exception is thrown

description:
the in keyword can be used to search for a value in a list, set, or dict. In this case the search fails, because the string value is case sensitive.

Question #46: **What gets printed?**

```
names1 = ['Amir', 'Barry', 'Chales', 'Dao']

loc = names1.index("Edward")

print(loc)
```

- -1
- 0
- 4
- Edward
- An exception is thrown - **correct**

description:
If index can not find the specified value in the list an exception is thrown.

Question #45: **What gets printed?**

```
names1 = ['Amir', 'Barry', 'Chales', 'Dao']
names2 = names1
names3 = names1[:]

names2[0] = 'Alice'
names3[1] = 'Bob'

sum = 0
for ls in (names1, names2, names3):
    if ls[0] == 'Alice':
        sum += 1
    if ls[1] == 'Bob':
        sum += 10

print(sum)
```

- 11
- 12 - **correct**
- 21
- 22
- 33

description:
When assigning names1 to names2, we create a second reference to the same list. Changes to names2 affect names1.

When assigning the slice of all elements in names1 to names3, we are creating a full copy of names1 which can be modified independently.

Question #44: **What gets printed?**

```
names = ['Amir', 'Barry', 'Chales', 'Dao']
print(names[-1][-1])
```

- A
- r
- Amir
- Dao
- o - **correct**

description:
-1 refers to the last position in a list or the last character in a string.

In this case, we are referencing the last character in the last string in the list.

Question #43: **What gets printed?**

```
foo = {1:'1', 2:'2', 3:'3'}
del foo[1]
foo[1] = '10'
del foo[2]
print(len(foo))
```

- 1
- 2 - **correct**
- 3
- 4
- An exception is thrown

description:
The del function is used to remove key value pairs from a dictionary.

Question #42: **What gets printed?**

```
foo = {1:'1', 2:'2', 3:'3'}
foo = {}
print(len(foo))
```

- ○ 0 - **correct**
- ○ 1
- ○ 2
- ○ 3
- ○ An exception is thrown

description:
after the second line of code, foo is an empty dictionary. The proper way to actually remove all items from a dictionary is to call the 'clear' method of the dictionary object

Question #41: **What gets printed?**

```
numberGames = {}
numberGames[(1,2,4)] = 8
numberGames[(4,2,1)] = 10
numberGames[(1,2)] = 12

sum = 0
for k in numberGames:
    sum += numberGames[k]

print(len(numberGames) + sum)
```

- ○ 8
- ○ 12
- ○ 24
- ○ 30
- ○ 33 - **correct**

description:
Tuples can be used for keys into dictionary. The tuples can have mixed length and the order of the items in the tuple is considered when comparing the equality of the keys.

Question #40: **What gets printed?**

```
boxes = {}
jars = {}
crates = {}

boxes['cereal'] = 1
boxes['candy'] = 2
jars['honey'] = 4
crates['boxes'] = boxes
crates['jars'] = jars

print(len(crates[boxes]))
```

- ○ 1
- ○ 2
- ○ 4
- ○ 7
- ○ An exception is thrown - **correct**

description:
Keys can only be immutable types, so a dictionary can not be used as a key. In the print statement the dictionary is used as the key instead of the string 'boxes'. Had the string been used it would have printed the length of the boxes dictionary which is 2.

Question #39: **What gets printed?**

```
confusion = {}
confusion[1] = 1
confusion['1'] = 2
confusion[1.0] = 4

sum = 0
for k in confusion:
    sum += confusion[k]

print(sum)
```

- ○ 2
- ○ 4
- ○ 6 - **correct**
- ○ 7
- ○ An exception is thrown

description:
Note from python docs: "if two numbers compare equal (such as 1 and 1.0) then they can be used interchangeably to index the same dictionary entry. (Note however, that since computers store floating-point numbers as approximations it is usually unwise to use them as dictionary keys.)"

Question #38: **What gets printed?**

```
confusion = {}
confusion[1] = 1
confusion['1'] = 2
confusion[1] += 1

sum = 0
for k in confusion:
    sum += confusion[k]

print(sum)
```

- 1
- 2
- 3
- 4 - **correct**
- 5

description:
Note that keys to a dictionary can be mixed between strings and integers and they represent different keys.

Question #37: **What gets printed?**

```
country_counter = {}

def addone(country):
    if country in country_counter:
        country_counter[country] += 1
    else:
        country_counter[country] = 1

addone('China')
addone('Japan')
addone('china')

print(len(country_counter))
```

- ○ 0
- ○ 1
- ○ 2
- ○ 3 - **correct**
- ○ 4

description:
The len function will return the number of keys in a dictionary. In this case 3 items have been added to the dictionary. Note that the key's to a dictionary are case sensitive.

Question #36: **What gets printed?**

```
foo = (3, 4, 5)
print(type(foo))
```

- ○ <class 'int'>
- ○ <class 'list'>
- ○ <class 'tuple'> - **correct**
- ○ <class 'dict'>
- ○ <class 'set'>

description:
Parentheses are used to initialize a tuple.

Question #35: **What gets printed**

```
foo = {}
print(type(foo))
```

- ○ <class 'set'>
- ○ <class 'dict'> - **correct**
- ○ <class 'list'>
- ○ <class 'tuple'>
- ○ <class 'object'>

description:
Curly braces are the syntax for a dictionary declaration

Question #34: **Assuming the filename for the code below is /usr/lib/python/person.py**
and the program is run as:
`python /usr/lib/python/person.py`

What gets printed?

```
class Person:
    def __init__(self):
        pass

    def getAge(self):
        print(__name__)

p = Person()
p.getAge()
```

- ○ Person
- ○ getAge
- ○ usr.lib.python.person
- ○ __main__ - **correct**
- ○ An exception is thrown

description:
If the module where the reference to __name__ is made has been imported from another file, then the module name will be in the variable in the form of the filename without the path or file extension. If the code is being run NOT as the result of an import, the variable will have the special value "__main__".

Question #33: **True or false? Code indentation must be 4 spaces when creating a code block?**

```
if error:
    # four spaces of indent are used to create the block
    print(msg)
```

- ○ True
- ○ False - **correct**

description:
This is false. Indentation needs to be consistent. A specific number of spaces used for indentation is not prescribed by the language.

Question #32: **Which of the following print statements will print all the names in the list on a seperate line**

```
names = ['Ramesh', 'Rajesh', 'Roger', 'Ivan', 'Nico']
```

- ○ print("\n".join(names)) - **correct**
- ○ print(names.join("\n"))
- ○ print(names.concatenate("\n"))
- ○ print(names.append("\n"))
- ○ print(names.join("%s\n", names))

description:
Only A is valid syntax.

There is a join method to string objects which takes an iterable object as parameter and combines the string calling the method in between each item to produce a resulting string.

Question #31: **What gets printed?**

```
import re
sum = 0

pattern = 'back'
if re.match(pattern, 'backup.txt'):
    sum += 1
if re.match(pattern, 'text.back'):
    sum += 2
if re.search(pattern, 'backup.txt'):
    sum += 4
if re.search(pattern, 'text.back'):
    sum += 8

print(sum)
```

- 3
- 7
- 13 - **correct**
- 14
- 15

description:
search will see if the pattern exists anywhere in the string, while match will only check if the pattern exists in the beginning of the string.

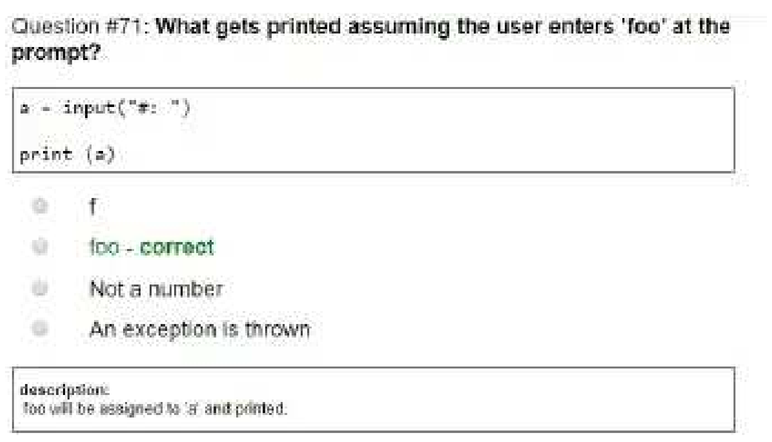
Question #30: **What does the code below do?**

```
sys.path.append('/root/mods')
```

- Changes the location that the python executable is run from
- Changes the current working directory
- Adds a new directory to seach for python modules that are imported - **correct**
- Removes all directories for mods
- Changes the location where sub-processes are searched for after they are launched

description:
The list sys.path contains, in order, all the directories to be searched when trying to load a module

DE LA 71 LA 80

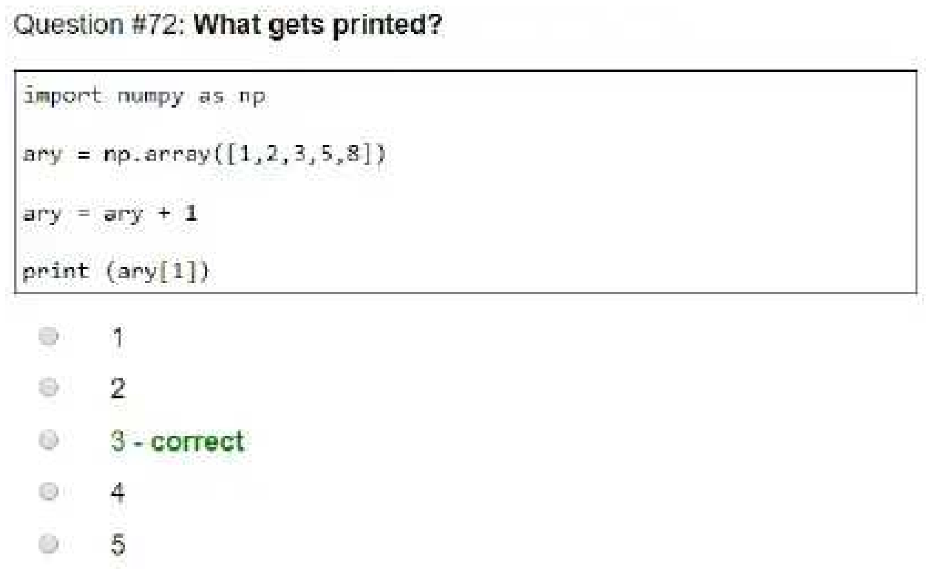


Question #71: **What gets printed assuming the user enters 'foo' at the prompt?**

```
a = input("#: ")

print (a)
```

- f
- foo - correct
- Not a number
- An exception is thrown

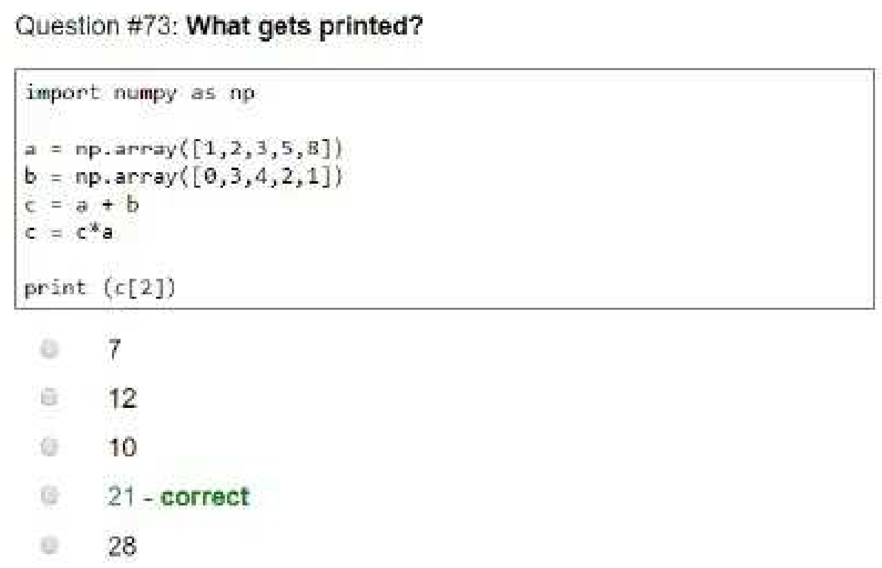
description:
foo will be assigned to 'a' and printed.



Question #72: **What gets printed?**

```
import numpy as np

ary = np.array([1,2,3,5,8])

ary = ary + 1

print (ary[1])
```

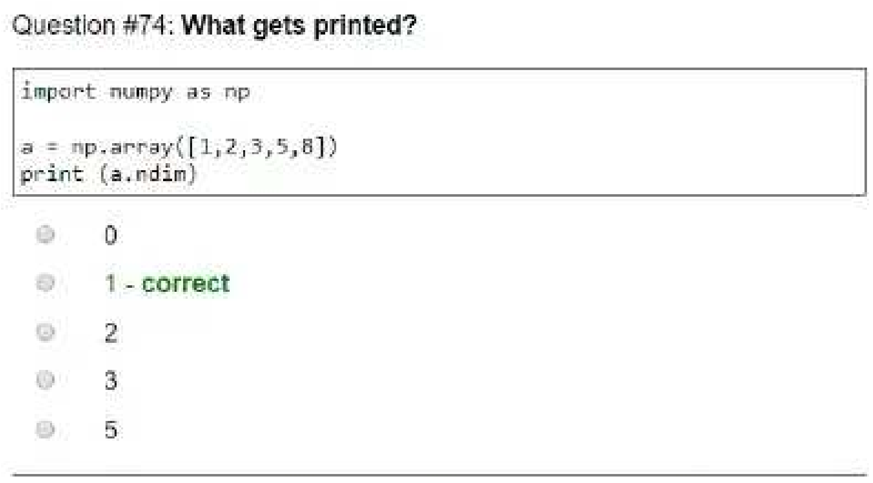
- 1
- 2
- 3 - correct
- 4
- 5



Question #73: **What gets printed?**

```
import numpy as np

a = np.array([1,2,3,5,8])
b = np.array([0,3,4,2,1])
c = a + b
c = c*a

print (c[2])
```

- 7
- 12
- 10
- 21 - correct
- 28



Question #74: **What gets printed?**

```
import numpy as np

a = np.array([1,2,3,5,8])
print (a.ndim)
```

- 0
- 1 - correct
- 2
- 3
- 5

Question #75: **What gets printed?**

```
import numpy as np

a = np.array([[1,2,3],[0,1,4]])
print (a.size)
```

- 0
- 1
- 2
- 5
- 6 - **correct**

Question #76: **What gets printed?**

```
import numpy as np

a = np.array([[1,2,3],[0,1,4]])
b = np.zeros((2,3), dtype=np.int16)
c = np.ones((2,3), dtype=np.int16)
d = a + b + c
print (d[1,2] )
```

- 0
- 1
- 5 - **correct**
- 8
- An exception is thrown

Question #77: **What gets printed?**

```
import numpy as np

a = np.array([1,2,3,4,5])
b = np.arange(0,10,2)
c = a + b
print (c[4])
```

- 4
- 5
- 13 - **correct**
- 15
- An exception is thrown

Question #78: **What gets printed?**

```
import numpy as np

a = np.zeros(6)
b = np.arange(0,10,2)
c = a + b
print (c[4])
```

- 0
- 2
- 8
- 10
- An exception is thrown - **correct**

Question #79: **What gets printed?**

```
import numpy as np

a = np.array([[0, 1, 0], [1, 0, 1]])
a += 3
b = a + 3
print (a[1,2] + b[1,2])
```

- 2
- 8
- 11 - **correct**
- 14
- An exception is thrown

Question #80: **What gets printed?**

```
import numpy as np

a = np.array([[0, 1, 2], [3, 4, 5]])
b = a.sum(axis=1)
print (b)
```

- 3
- 12
- [3 12] - **correct**
- [3 5 7]
- An Exception is thrown

Question #81: **What gets printed?**

```
import numpy as np

a = np.array([[0, 1, 2], [3, 4, 5]])
b = a.ravel()
print (b[0,0])
```

- 0
- 3
- 9
- 10
- An exception is thrown - **correct**