Exercises

Answer the question below, then and check your responses using the **Python REPL or creating and executing a .py file.**

1) Check all options that contains keywords used in loop control?

a) `break` ( )
b) `stop` ( )
c) `else` ( )
d) `continue` ( )
e) `in` ( )

2) What command should be used to exit this while loop?

```
while True:
  print(“Hi”)
```

a) `break`
b) `stop`
c) `terminate`
d) `continue`

3) What’s the output of the example below?

```
i = 1
while True:
  print(i)
  i += 1
  if i == 42:
    break

print(i)
```

a) 40
b) 41
c) 42
d) 43

4) What’s the output of the example below?

```
s = “acbdefgh”
m = “”
for c in s:
  if c in “aeiou”:
    continue
  m += c
print(m)
```

a) `abcdefgh`
b) `ae`
c) `cbdfgh`
d) `fgh`

5) What's the output of the example below?

```
c = 0
while c < 1:
  c += 1
  print(c)
  if c == 1:
    break
else:
  print(“Else executed!”)
```

a) `1`
b) `1`
   `Else executed`
c) `Else executed`
d) `Nothing is printed`

6) What's the output of the example below?

```
i = []
for v in i:
  print(“Iterating...”)
else:
  print(“Iteration over!”)
```

a) `Iterating...`
   `Iteration over!`
b) `Iterating...`
c) `Iteration over!`
d) `Nothing is printed`

7) Are the lists comprehensions defined below valid? Mark as True of False.

```
a) lc = [a for b in range(5)]                           (    )
b) lc = [a for a in range(5) if a < 2]                  (    )
c) lc = [b**2 for b in range(3)]                        (    )
d) lc = [[(a, b) for a in range(5)] for b in range(3)]  (    )
```

8) What is the equivalent of this for loop as a list comprehension?

```
sqrt = []
for v in range(5, 10):
  sqrt.append(v ** .5)
```

a) `sqrt = [v ** .5 for v in range(5)]`
b) `sqrt = [v ** 2 for v in range(5, 10)]`
c) `sqrt = [c ** .5 for v in range(5)]`
d) `sqrt = [v ** .5 for v in range(5, 10)]`

9) What is the equivalent of this for loop as a dict comprehension?

```
dc = {}
for c in (1, 2, 3, 4, 5):
  dc[c] = c ** 2
```

a) `dc = {c: c ** 2 for c in range(5)}`
b) `dc = {c: c ** .5 for c in range(1, 6)}`
c) `dc = {c: c ** 2 for c in range(1, 5)}`
d) `dc = {c: c ** 2 for c in range(1, 6)}`

10) What is the equivalent of this for loop as a set comprehension?

```
sc = {}
for c in “abracadabra”:
  if c != “a”:
    sc.add(c)
```

a) `sc = [c for c in “abracadabra”]`
b) `sc = [c for c in “abracadabra” if c == “a”]`
c) `sc = [c for c in “abracadabra” if c != “a”]`
d) `sc = [c for c in “brcdbrf”]`

11) Consider the sets defined below, what are the results of the operations?

```
s1 = {1, 2, “B”, 4, 5}
s2 = {1, “A”, “B”, “C”, 5}
```

a) `>>> s1 & s2`
b) `>>> s1 ^ s2`
c) `>>> s2 | s2`
d) `>>> s1 – s2`
e) `>>> s2 – s1`
f) `>>> s2 & s1`
g) `>>> s2 ^ s1`

12) Given the function definition, what are the results of the function calls below?

```
def mult_values(a, b, c=.5, *args, **kwargs):
  result = a * b
  result *= c
  for v in args:
    result *= v
  for k, v in kwargs.items():
    result *= v
  print(args)
  print(kwargs)
  return result
```

a) `>>> mult_values(1, 2, d=2)`

b) `>>> mult_values(b=2, a=1, d=1, f=2)`

c) `>>> mult_values(1, 2, 3, 2, 2)`

d) `>>> mult_values(1, 2, 3, 2, 2, house=2)`

e) `>>> mult_values(1, 2, 1, 1, hi=2, hello=2, greetings=1)`

f)
```
>>> a = [1, 2, 3, 4, 5]
>>> mult_values(*a)
```

g)
```
>>> d = {“a”:1, “b”:2, “d”:3}
>>> mult_values(**d)
```

h)
```
>>> d = dict(j=1, k=2)
>>> a = [1, 2, 3, 4]
>>> mult_values(1, 2, *a, **d)
```

13) Are the Lambda Expressions defined below valid? Mark as True of False.

a) `lambda x ** 2: x` ( )
b) `x lambda : x ** 2` ( )
c) `lambda b: b + 1` ( )
d) `lambda x: x **2` ( )
e) `lambda a: b ** 2` ( )

14) Given the class definition, what are the results of the alternatives below?

```
class Circle:
  pi = 3.14
  def __init__(self, radius):
    self.radius = radius

  def area(self):
    a = self.pi * r ** 2
    print('Area:' + str(a))

  def circumference(self):
    c = self.pi * r * 2
    print(' Circumference:' + str(c))
```

a)
```
>>> c = Circle(2)
>>> print(c.radius)
```

b)
```
>>> c = Circle(2)
>>> c.radius = 3
>>> print(c.radius)
```

c)
```
>>> c = Circle((1/3.14)**2)
>>> c.area()
```

d)
```
>>> c = Circle(1/3.14)
>>> c.circumference()
```

e)
```
>>> c1 = Circle(1)
>>> c2 = Circle(2)
>>> c1.radius * c2.radius
```

f)
```
>>> c1 = Circle(1)
>>> c2 = Circle(2)
>>> c1.pi / c2.pi
```

g)
```
>>> Circle.pi = 1
>>> c = Circle(2)
>>> c.area()
```

15) Consider the generic module.py below. Which of the alternatives are valid import statements? Mark as True of False.

```
class Circle:
  pi = 3.14
  def __init__(self, radius):
    self.radius = radius

  def area(self):
    a = self.pi * r ** 2
    print('Area:' + str(a))

  def circumference(self):
    c = self.pi * r * 2
    print('Circumference:' + str(c))

def add_area(c1, c2)
  return c1.area() + c2.area()

if __name__ == "__main__":
    c = Circle(1/3.14)
    c.area()
```

a) `import module.py` (   )
b) `in module import Circle` (   )
c) `from module import add_area` (   )
d) `from module import *` (   )
e) `import module as c` (   )
f) `from module Circle as C` (   )
g) `from module import Circle as C` (   )

16) What is printed when the module in the previous question is executed?