

```
def get_jugs():
```

Returns a list of two integers representing volumes of the jugs.

Takes volumes of the jugs as an input from the user.

```
    """
```

```
    print("Receiving the volume of the jugs...")
```

```
    jugs = []
```

```
    temp = int(input("Enter first jug volume (>1):"))
```

```
    while temp < 1:
```

```
        temp = int(input("Enter a valid amount (>1): "))
```

```
        jugs.append(temp)
```

```
    temp = int(input("Enter second jug volume (>1):"))
```

```
    while temp < 1:
```

```
        temp = int(input("Enter a valid amount (>1): "))
```

```
        jugs.append(temp)
```

```
    return jugs
```

```
def get_goal(jugs):
```

Returns desired amount of water.

Takes desired amount as an input from the user.

```
jugs: a list of two integers representing  
volumes of the jugs
```

```
print("Receiving the desired amount of the  
water...")
```

```
max_amount = max(jugs[0], jugs[1])
```

```
s = "Enter the desired amount of water (1 - {0}):".format(max_amount)
```

```
goal_amount = int(input(s))
```

```
while goal_amount < 1 or goal_amount >  
max_amount:
```

```
goal_amount = int(input("Enter a valid amount  
(1 - {0}): ".format(max_amount)))
```

```
return goal_amount
```

def is_goal(path, goal_amount):

Returns True, if the given path terminates at the goal node.

path: a list of nodes representing the path to be checked

goal_amount: an integer representing the desired amount of water

print("Checking if the goal is achieved...")

return path[-1][0] == goal_amount or path[-1][1] == goal_amount

def been_there(node, check_dict):

Returns True, if the given node is already visited

node: a list of two integers representing current state of the jugs

check_dict: a dictionary storing visited nodes

....
print("Checking if {} is visited
before...".format(node))
return check_dict.get(get_index(node), False)

def next_transitions(jugs, path, check_dict):

....
Returns list of all possible transitions which
do not cause loops

jugs: a list of two integers representing
volumes of the jugs

path: a list of nodes representing the current
path

check_dict: a dictionary storing visited nodes

....
print("Finding next transitions and checking for
the loops...")

result = []

next_nodes = []

node = []

$a_{max} = jugs[0]$
 $b_{max} = jugs[1]$

$a = path[-1][0] \#$ initial amount in the first jug
 $b = path[-1][1] \#$ initial amount in the second jug

1. fill in the first jug

node.append(a_{max})
node.append(b)
if not been there(node, check_dict):
 next_nodes.append(node)
node = []

2. fill in the second jug

node.append(a)
node.append(b_{max})
if not been there(node, check_dict):
 next_nodes.append(node)
node = []

3. second jug to first jug

node.append(min(a_{max} , $a + b$))
node.append($b - (node[0] - a)$) # $b - (a - a)$
if not been there(node, check_dict):

```
next_nodes.append(node)
node = []
```

```
# 4. first jug to second jug
node.append(min(a + b, b_max))
node.insert(0, a - (node[0] - b))
if not been_there(node, check_dict):
    next_nodes.append(node)
node = []
```

```
# 5. empty first jug
node.append(0)
node.append(b)
if not been_there(node, check_dict):
    next_nodes.append(node)
node = []
```

```
# 6. empty second jug
node.append(a)
node.append(0)
if not been_there(node, check_dict):
    next_nodes.append(node)
```

```
# create a list of next paths
for i in range(0, len(next_nodes)):
```

```
temp = list(path)
temp.append(next_nodes[i])
result.append(temp)
```

```
if len(next_nodes) == 0:
    print("No more unvisited
    nodes... In Backtracking...")
else:
    print("Possible transitions: ")
    for nnode in next_nodes:
        print(nnode)

return result
```

```
def transition(old, new, jugs):
```

returns a string explaining the transition from
old state/node to new state/node

old: a list representing old state/node

new: a list representing new state/node

jugs: a list of two integers representing
volumes of the jugs

$a = \text{old}[0]$

$b = \text{old}[1]$

$a_{\text{prime}} = \text{new}[0]$

$b_{\text{prime}} = \text{new}[1]$

$a_{\text{max}} = \text{jugs}[0]$

$b_{\text{max}} = \text{jugs}[1]$

if $a > a_{\text{prime}}$:

if $b == b_{\text{prime}}$:

return 'Clear {0}-liter

jug: |t|t|t|.format(a_max)

else:

return 'Pour {0}-liter jug into {1}-liter

jug: |t|.format(a_max, b_max)

else:

if $b > b_{\text{prime}}$:

if $a == a_{\text{prime}}$:

return 'Clear {0}-liter

jug: |t|t|t|.format(b_max)

else:

return 'Pour {0}-liter jug into {1}-liter

jug: |t|.format(b_max, a_max)

else:

if $a == a_{\text{prime}}$:

```
return "Fill {0}liter jug: |{1}|{2}|{3}|".format(b, max)
else:
    return "Fill {0}liter jug: |{1}|{2}|{3}|".format(a, max)
```

def print_path(path, jugs):

"""
 prints the goal path

path: a list of nodes representing the goal
 path

jugs: a list of two integers representing
 volumes of the jugs

"""

```
print("Starting from: |{0}|{1}|{2}|", path[0])
for i in range(1, len(path) - 1):
    print(i+1, ":", transition(path[i], path[i+1], jugs),
          path[i+1])
```