

implementationArbre

```
In [1]: class Arbre:
    def __init__(self, etiquette):
        self.etiquette = etiquette
        self.gauche = None
        self.droit = None

    def ajout_gauche(self, sousarbre): # mutateur
        self.gauche = sousarbre

    def ajout_droit(self, sousarbre): # mutateur
        self.droit = sousarbre

    def get_gauche(self): # accesseur
        return self.gauche

    def get_droit(self): # accesseur
        return self.droit

    def get_etiquette(self): # accesseur
        return self.etiquette

    def affiche(self):
        """permet d'afficher un arbre"""
        if self==None:
            return None
        else :
            return
[ self.etiquette, Arbre.affiche(self.gauche), Arbre.affiche(self.droit) ]
```

```
In [3]: a = Arbre(4)
a.ajout_gauche(Arbre(3))
a.ajout_droit(Arbre(1))
a.get_droit().ajout_gauche(Arbre(2))
a.get_droit().ajout_droit(Arbre(7))
a.get_gauche().ajout_gauche(Arbre(6))
a.get_droit().get_droit().ajout_gauche(Arbre(9))
a.affiche()
```

```
Out[3]: [4,
        [3, [6, None, None], None],
        [1, [2, None, None], [7, [9, None, None], None]]]
```

```
In [5]: print(a.get_droit().get_gauche().get_etiquette())
```

2

```
In [6]: class Arbre:
    def __init__(self, valeur):
        """Initialisation de l'arbre racine + sous-arbre gauche et sous-arbre
        droit"""
        self.v=valeur
        self.gauche=None
        self.droit=None

    def ajout_gauche(self, val):
        """On ajoute valeur dans le sous-arbre gauche sous la forme
        [val, None, None]"""
        self.gauche=Arbre(val)

    def ajout_droit(self, val):
        """ On ajoute valeur dans le sous-arbre droit sous la forme
        [val, None, None]"""
        self.droit=Arbre(val)

    def get_gauche(self):
        return self.gauche

    def get_droit(self):
        return self.droit

    def get_valeur(self):
        if self==None:
            return None
        else:
            return self.v

    def affiche(self):
        """permet d'afficher un arbre"""
        if self==None:
            return None
        else :
            return
            [self.v, Arbre.affiche(self.gauche), Arbre.affiche(self.droit)]
```

```
In [7]: a = Arbre(4)
a.ajout_gauche(3)
a.ajout_droit(1)
a.droit.ajout_gauche(2)
a.droit.ajout_droit(7)
a.gauche.ajout_gauche(6)
a.droit.droit.ajout_gauche(9)
print(a.affiche())
a.get_droit().affiche()
```

```
[4, [3, [6, None, None], None], [1, [2, None, None], [7, [9, None, None], None]]]
```

```
Out[7]: [1, [2, None, None], [7, [9, None, None], None]]
```

```
In [72]: class Arbre:
    def __init__(self, etiquette):
        self.etiquette = etiquette
        self.gauche = None
        self.droit = None

    def est_feuille(self):
        if self.gauche==None and self.droit==None:
            return True
        else:
            return False

    def affiche(self):
        """permet d'afficher un arbre"""
        if self==None:
            return None
        else :
            return
[self.etiquette,Arbre.affiche(self.gauche),Arbre.affiche(self.droit)]
```

```
In [91]: class Arbre:
    def __init__(self, etiquette,gauche=None,droit=None):
        self.etiquette = etiquette
        self.gauche = None
        self.droit = None

    def est_feuille(self):
        if self.gauche==None and self.droit==None:
            return True
        else:
            return False

    def arbre_digraph(self):
        noeuds=[]
        aretes=[]
        if self!=None:
            noeuds=[self.etiquette]
            if self.gauche!=None:
                aretes.append([self.etiquette,self.gauche.etiquette])
                sag = Arbre(self.gauche)
                pg = sag.arbre_digraph()
                noeuds = noeuds + pg[0]
                aretes = aretes + pg[1]
            if self.droit!=None:
                aretes.append([self.etiquette,self.droit.etiquette])
                sad = Arbre(self.droit)
                pd = sad.arbre_digraph()
                noeuds = noeuds + pd[0]
                aretes = aretes + pd[1]
            return noeuds,aretes

    def affiche(self):
        # création de l'objet graphviz qui sera renvoyé
        img_arbre = Digraph()
```

```

    noeuds, aretes = self.arbre_digraph()
    for n in noeuds:
        img_arbre.node(n,n)
    for a in aretes:
        img_arbre.edge(a[0],a[1])
    return img_arbre

```

In [92]:

```

a = Arbre(4)
a.gauche = Arbre(3)
a.droit = Arbre(1)
a.droit.gauche = Arbre(2)
a.droit.droit = Arbre(7)
a.gauche.gauche = Arbre(6)
a.droit.droit.gauche = Arbre(9)

```

In [93]:

```

a.affiche()

```

```

-----
TypeError                                Traceback (most recent call last)
Cell In [93], line 1
----> 1 a.affiche()

Cell In [91], line 37, in Arbre.affiche(self)
    35 noeuds, aretes = self.arbre_digraph()
    36 for n in noeuds:
--> 37     img_arbre.node(n,n)
    38 for a in aretes:
    39     img_arbre.edge(a[0],a[1])

File ~/.local/lib/python3.10/site-packages/graphviz/_tools.py:171, in deprecate_positional_args.<locals>.decorator.<locals>.wrapper(*args, **kwargs)
    162     wanted = ', '.join(f'{name}={value!r}'
    163                        for name, value in deprecated.items())
    164     warnings.warn(f'The signature of {func.__name__} will be reduced'
    165                  f' to {supported_number} positional args'
    166                  f' {list(supported)}: pass {wanted}'
    167                  ' as keyword arg(s)',
    168                  stacklevel=stacklevel,
    169                  category=category)
--> 171 return func(*args, **kwargs)

File ~/.local/lib/python3.10/site-packages/graphviz/dot.py:195, in Dot.node(self, f, name, label, _attributes, **attrs)
    184 @_tools.deprecate_positional_args(supported_number=3)
    185 def node(self, name: str,
    186         label: typing.Optional[str] = None,
    187         _attributes=None, **attrs) -> None:
    188     """Create a node.
    189
    190     Args:
    (...)
    193     attrs: Any additional node attributes (must be strings).
    194     """

```

```
--> 195     name = self._quote(name)
      196     attr_list = self._attr_list(label, kwargs=attrs, attributes=_attribu
tes)
      197     line = self._node(name, attr_list)
```

File ~/.local/lib/python3.10/site-packages/graphviz/_tools.py:171, in deprecate_positional_args.<locals>.decorator.<locals>.wrapper(*args, **kwargs)

```
      162     wanted = ', '.join(f'{name}={value!r}'
      163                           for name, value in deprecated.items())
      164     warnings.warn(f'The signature of {func.__name__} will be reduced'
      165                  f' to {supported_number} positional args'
      166                  f' {list(supported)}: pass {wanted}'
      167                  ' as keyword arg(s)',
      168                  stacklevel=stacklevel,
      169                  category=category)
--> 171 return func(*args, **kwargs)
```

File ~/.local/lib/python3.10/site-packages/graphviz/quotting.py:82, in quote(identifier, is_html_string, is_valid_id, dot_keywords, ends_with_odd_number_of_backslashes, escape_unescaped_quotes)

```
      40 @_tools.deprecate_positional_args(supported_number=1)
      41 def quote(identifier: str,
      42           is_html_string=HTML_STRING.match,
      43           (...),
      44           ends_with_odd_number_of_backslashes=FINAL_ODD_BACKSLASHES.search,
      45           h,
      46           escape_unescaped_quotes=ESCAPE_UNESCAPED_QUOTES) -> str:
      47     r"""Return DOT identifier from string, quote if needed.
      48
      49     >>> quote('') # doctest: +NO_EXE
      50     ''
      51     """
--> 82     if is_html_string(identifier) and not isinstance(identifier, NoHtml
1):
      83         pass
      84     elif not is_valid_id(identifier) or identifier.lower() in dot_keywor
ds:
```

TypeError: expected string or bytes-like object

In [94]: a.droit.gauche.etiquette



Out[94]: 2

In [95]: a.droit.gauche.est_feuille()



Out[95]: True

In [96]: a.droit.droit.gauche.est_feuille()



Out[96]: True

```
In [97]: a.droit.gauche.est_feuille()
```

```
Out[97]: True
```

```
In [ ]: class Noeud:
        def __init__(self, etiquette, gauche=None, droit=None):
            self.etiquette = etiquette
            self.gauche = None
            self.droit = None
```

```
In [66]: class ArbreBinaire:

        def __init__(self, racine):
            # racine est soit None (arbre vide) soit un objet de la classe noeud
            self.racine=racine

        def affiche(self):
            """permet d'afficher un arbre"""
            if self==None:
                return None
            else :
                return
            [self.racine.etiquette, ArbreBinaire.affiche(self.racine.gauche), ArbreBinaire.a
```

```
In [67]: C = Noeud("C")
        D = Noeud("D")
        F = Noeud("F")
```

```
In [68]: # On peut à présent "remonter" dans l'arbre et déclarer les noeuds parents
        E = Noeud("E", F)
        B = Noeud("B", D, E)
```

```
In [69]: A=Noeud('A', B, C)
```

```
In [70]: A1=ArbreBinaire(A)
```

```
In [71]: print(A1.affiche())
```

```
['A', None, None]
```

```
In [ ]:
```