



WERTENTWICKLUNG IN DER VERGANGENHEIT*

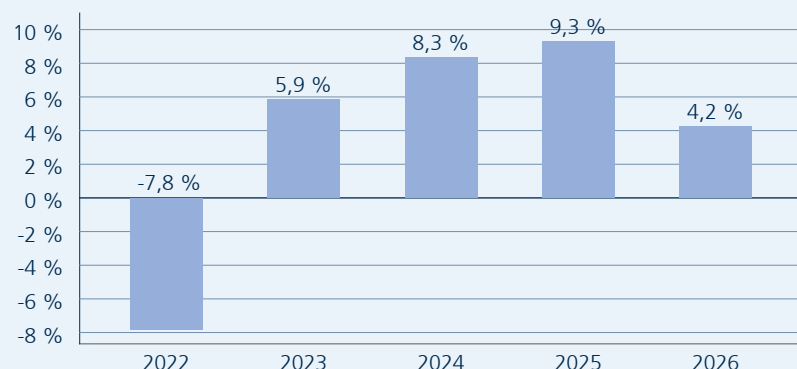
Indexierte Wertentwicklung



WERTENTWICKLUNG ÜBER VERSCHIEDENE ZEITRÄUME

1 Monat	3 Monate	6 Monate	lfd. Jahr	1 Jahr
0,80 %	5,58 %	9,29 %	4,24 %	11,44 %
3 Jahre p.a.	5 Jahre p.a.	10 Jahre p.a.	Seit Auflage p.a.	
8,71 %	5,09 %	-	3,32 %	
3 Jahre	5 Jahre	10 Jahre	Seit Auflage	
28,52 %	28,21 %	-	34,82 %	

WERTENTWICKLUNG PRO KALENDERJAHR



ROLLIERENDE 12-MONATS ENTWICKLUNG

23.02.22-23.02.23	23.02.23-23.02.24	23.02.24-23.02.25	23.02.25-23.02.26
-3,20 %	6,08 %	8,72 %	11,44 %

*Historische Wertentwicklungen sind keine Garantie für eine ähnliche Entwicklung in der Zukunft. Diese ist nicht prognostizierbar. Die Berechnung der Wertentwicklung erfolgt gemäß BVI-Methode.

23. FEBRUAR 2026

Anlageziel

Das Fondsvermögen kann in Aktien, Anleihen (auch hochverzinsliche Anlagen, Contingent Convertible Bonds und forderungsbesicherte Wertpapiere), Geldmarktinstrumente, Rohstoffe und Währungen angelegt werden. Die Anlagen können über Direktinvestments (außer Rohstoffe) und in Form von Verbriefungen beziehungsweise Derivaten sowie Zielfonds erfolgen. Das Anlagekonzept des Fonds verbindet ein aktives Management mit einer flexiblen Steuerung einer Vielzahl von Anlageklassen. Dabei wird angestrebt, dass das Schwankungsverhalten des Anteilwertes weitgehend unabhängig von der Entwicklung des Schwankungsverhaltens einzelner Anlageklassen ist und der Fonds insgesamt defensiver ausgerichtet wird. Die langfristige Schwankungsbreite des Fonds orientiert sich dabei am durchschnittlichen Schwankungsverhalten eines defensiven Mischportfolios, welches aus circa 35 Prozent Aktien und 65 Prozent Renten zusammengesetzt ist. Die tatsächliche Fondszusammensetzung bemisst sich dabei am jeweils geschätzten Schwankungsverhalten der einzelnen eingesetzten Anlageklassen, wobei die Aktiengewichtung in der Regel zwischen 10 und 50 Prozent beträgt. Die Anlagestrategie orientiert sich nicht an einem Vergleichsmaßstab, sondern versucht das Renditeziel zu erreichen/ zu übertreffen. Das Fondsmanagement trifft dabei auf Basis aktueller Kapitalmarktinschätzungen aktive Anlageentscheidungen.

```

class MirrorX(object):
    """This adds an
    "ob"
    "Mir"
    "Cont"

    def __init__(self, mirror_mod, mirror_obj):
        self.mirror_mod = mirror_mod
        self.mirror_obj = mirror_obj

    def __getattr__(self, attr):
        if attr == "x":
            return self.mirror_mod.x
        elif attr == "y":
            return self.mirror_mod.y
        elif attr == "z":
            return self.mirror_mod.z
        else:
            return self.mirror_obj.__getattr__(attr)

    def __setattr__(self, attr, value):
        if attr == "x":
            self.mirror_mod.x = value
        elif attr == "y":
            self.mirror_mod.y = value
        elif attr == "z":
            self.mirror_mod.z = value
        else:
            self.mirror_obj.__setattr__(attr, value)

    def __delattr__(self, attr):
        if attr == "x":
            del self.mirror_mod.x
        elif attr == "y":
            del self.mirror_mod.y
        elif attr == "z":
            del self.mirror_mod.z
        else:
            del self.mirror_obj.__delattr__(attr)

    def __str__(self):
        return str(self.mirror_obj)

    def __repr__(self):
        return repr(self.mirror_obj)

    def __eq__(self, other):
        return self.mirror_obj == other

    def __neq__(self, other):
        return self.mirror_obj != other

    def __lt__(self, other):
        return self.mirror_obj < other

    def __gt__(self, other):
        return self.mirror_obj > other

    def __le__(self, other):
        return self.mirror_obj <= other

    def __ge__(self, other):
        return self.mirror_obj >= other

    def __add__(self, other):
        return self.mirror_obj + other

    def __sub__(self, other):
        return self.mirror_obj - other

    def __mul__(self, other):
        return self.mirror_obj * other

    def __div__(self, other):
        return self.mirror_obj / other

    def __mod__(self, other):
        return self.mirror_obj % other

    def __pow__(self, other):
        return self.mirror_obj ** other

    def __call__(self):
        return self.mirror_obj

    def __getitem__(self, item):
        return self.mirror_obj[item]

    def __setitem__(self, item, value):
        self.mirror_obj[item] = value

    def __delitem__(self, item):
        del self.mirror_obj[item]

    def __iter__(self):
        return iter(self.mirror_obj)

    def __len__(self):
        return len(self.mirror_obj)

    def __contains__(self, item):
        return item in self.mirror_obj

    def __getitem__(self, item):
        return self.mirror_obj[item]

    def __setitem__(self, item, value):
        self.mirror_obj[item] = value

    def __delitem__(self, item):
        del self.mirror_obj[item]

    def __iter__(self):
        return iter(self.mirror_obj)

    def __len__(self):
        return len(self.mirror_obj)

    def __contains__(self, item):
        return item in self.mirror_obj
    
```

