Cheatography

Pkl Cheat Sheet Cheat Sheet

by lis186 via cheatography.com/200657/cs/42381/

Comments

Line comment	// Single-line comment
Block comment	/* First line Second line */
Doc comment	/// A *bird* superstar. /// Unfortunately, extinct. dodo: Bird

Integers

decimal	123	123
hexadecimal	0x012AFF	76543
binary	0b00010111	23
octal	0o755	493
decimal with underscore	1_000_000	1000000
hexadecimal with underscore	0x0134_64DE	0x013464DE
binary with underscore	0b0001_0111	0b00010111
octal with underscore	0o0134_6475	0o01346475
To restrict an integer's range	clientPort: UInt16	
To restrict an integer's range	serverPort: Int(isBetween(0, 1023))	

Comparison operators for integers

5 == 2	false
5 < 2	false
5 > 2	true
5 <= 2	false
5 >= 2	true

Arithmetic operators for integers

5 + 2	7	
5 - 2	3	
5 * 2	10	
5 / 2	2.5	always Float
5 ~/ 2	2	always Int
5 % 2	1	always Int
5 ** 2	25	

Floats

.23	0.23
1.23	1.23
1.23e2	123
1.23e-2	0.0123
NaN	not a number
Infinity	positive infinity
-Infinity	negative infinity
To Restrict a float to a finite value	x: Float(isFinite)
To restrict a float's range	y: Float(isBetween(0, 10e6))

Booleans

Boolean	true or false

Logical operators for Booleans

logical conjunction	true && false	false
logical disjunction	true || false	true
logical negation	!false	true
exclusive disjunction	true.xor(false)	true
logical implication	true.implies(false)	false

Pkl Cheat Sheet Cheat Sheet

by lis186 via cheatography.com/200657/cs/42381/

Strings

String	"Hello, World!"
Multiline string	""" Although the Dodo is extinct, the species will be remembered. """
Unicode	"\u{26} \u{E9} \u{1F600}"
To concatenate strings	"abc" + "def" + "ghi"
String Interpolation	name = "Dodo" greeting = "Hi, \(name)!"
Custom String Delimiters	#" \\\\\ """""" "#

Inside a string literal, the following character escape sequences have special meaning:

\t - tab
\n - line feed
\r - carriage return
\" - verbatim quote
\\ - verbatim backslash

String API

"dodo".length	4
"dodo".reverse()	"odod"
"dodo".contains("alive")	false
" dodo ".trim()	"dodo"

Dynamic objects vs.Typed Objects

Dynamic object	Schema-less data models that are not validated
Typed object	Schema-backed data models that are validated

Objects

Object with properties

```
dodo {
  name = "Dodo"
  extinct = true
}
```

Objects (cont)

Nested object

```
dodo {
  name = "Dodo"
  taxonomy {
    `class` = "Aves"
  }
}
```

An object is an ordered collection of values indexed by name.
An object's key–value pairs are called its properties.
Property values are lazily evaluated on the first read.

Typed Objects

```
class Bird {
    name: String
    lif espan: Int
    mig ratory: Boolean
}
pigeon = new Bird {
   name = " Pig eon "
    lif espan = 8
    mig ratory = false
}
```

Amending Objects

```
pigeon {
   name = " Turtle dove"
    extinct = false
}
parrot = (pigeon) {
   name = " Par rot "
}
```

Transforming Objects

```
dodo {
   name = " Dod o"
    extinct = true
}
dodo
   .to Map()
   .re mov e("n ame ")
   .to Dyn amic()
```

Converting untyped objects to typed objects

```
class Bird {
    name: String
    lif espan: Int
}
pigeon = new Dynamic {
   name = " Pig eon "
    lif espan = 8
}.toTy ped (Bird)
```

Late Binding

```
penguin {
    egg Inc ubation = 40.d
    adu ltW eig htI nGrams = eggInc uba tio n.value
* 100
}
adultW eig htI nGrams = pengui n.a dul tWe igh -
tIn Grams
madeUpBird = (penguin) {
    egg Inc ubation = 11.d
}
adultW eig htI nGrams = madeUp Bir d.a dul tWe -
igh tIn Grams
```

Hidden Properties

```
class Bird {
    name: String
    lif espan: Int
    hidden nameAn dLi fes pan InIndex = " \(n -
ame), \(life spa n)"
    nam eSi gnW idth: UInt = nameAn dLi fes pan -
InI nde x.l ength
}
/*
pigeon {
   name = " Pig eon "
    lif espan = 8
    nam eSi gnWidth = 9
}
*/
pigeon = new Bird {
   name = " Pig eon "
    lif espan = 8
}
pigeon InIndex = pigeon.na meA ndL ife spa nIn -
Index
// pigeon InIndex = " Pigeon, 8"
pigeon Dynamic = pigeon.to Dyn amic()
/*
pigeon Dynamic {
   name = " Pig eon "
    lif espan = 8
    nam eSi gnWidth = 9
}
*/
```

Local properties

```
class Bird {
    name: String
    lif espan: Int
    local separator = " ,"
    hidden nameAn dLi fes pan InIndex = " \(n -
ame )\( sep arator) \(life spa n)"
```

Local properties (cont)

```
> }
pigeon = new Bird {
  name = "Pigeon"
  lifespan = 8
}
pigeonInIndex = pigeon.nameAndLifespanInIndex
pigeonSeparator = pigeon.separator // Error
```

Fixed properties

```
class Bird {
    win gspan: Int
    weight: Int
    fixed wingsp anW eig htR atio: Float = wingspan
/ weight
}
bird = (Bird) {
    win gspan = 10
    weight = 8
}
```

A property with the fixed modifier cannot be assigned to or amended when defining an object of its class.

Extending a class with fixed properties

```
abstract class Bird {
    fixed canFly: Boolean
    name: String
}
class Penguin extends Bird {
    canFly = false // Error: missing modifier
fixed.
    fixed name = " Pen gui n" // Error: modifier
fixed cannot be applied to property name.
}
```

Const properties

```
const laysEggs: Boolean = true
const examples: Listin g<S tri ng> = new {
    " Pig eon "
    " Haw k"
    " Pen gui n"
}
pigeon Name: String = " Pig eon "
const function birdLi fes pan(i: Int): Int = (i /
4).toInt()
class Bird {
    name: String
    lif espan: Int
}
const bird: Bird = new {
   name = pigeonName // Error: cannot reference
non-const property pigeonName from a const
property.
    lif espan = birdLi fes pan(24) // Allowed:
birdLi fespan is const.
}
```

Because const members can only reference its own values, or other const members, they are not late bound.

Durations

nanoseconds (smallest unit)	5.ns
microseconds	5.us
milliseconds	5.ms
seconds	5.s
minutes	5.min
hours	5.h
days (largest unit)	3.d
negative duration	-5.min
duration with floating point value	5.13.min

Cheatography

Pkl Cheat Sheet Cheat Sheet

by lis186 via cheatography.com/200657/cs/42381/

Comparison operators for durations

5.min == 3.s	false
5.min < 3.s	false
5.min > 3.s	true
5.min <= 3.s	false
5.min >= 3.s	true

Arithmetic operators for durations

5.min + 3.s	5.05.min
5.min - 3.s	4.95.min
5.min * 3	15.min
5.min / 3	1.6666666666666667.min
5.min / 3.min	1.6666666666666667
5.min ~/ 3	1.min
5.min ~/ 3.min	1
5.min % 3	2.min
5.min ** 3	125.min

The value component can be an expression:

```
x = 5
xMinutes = x.min
y = 3
xySeconds = (x + y).s
```

Data sizes with decimal unit (factor 1000)

bytes (smallest unit)	5.b
kilobytes	5.kb
megabytes	5.mb
gigabytes	5.gb
terabytes	5.tb
petabytes (largest unit)	5.pb

Data sizes with binary unit (factor 1024)

bytes (smallest unit)	5.b
kilobytes	5.kb
megabytes	5.mb
gigabytes	5.gb
terabytes	5.tb
petabytes (largest unit)	5.pb

Arithmetic operators for data sizes

5.mb + 3.kib	5.003072.mb
5.mb - 3.kib	4.996928.mb
5.mb * 3	15.mb
5.mb / 3	1.6666666666666667.mb
5.mb / 3.mb	1.6666666666666667
5.mb ~/ 3	1.mb
5.mb ~/ 3.mb	1
5.mb % 3	2.mb
5.mb ** 3	125.mb

The value component can be an expression:

```
x = 5
xMegabytes = x.mb
y = 3
xyKibibytes = (x + y).kib
```

Comparison operators for data sizes

5.mb == 3.kib	false
5.mb < 3.kib	false
5.mb > 3.kib	true
5.mb <= 3.kib	false
5.mb >= 3.kib	true