class Allomorph

Dual value number and string

```
class Allomorph is Str { }
```

The `Allomorph` class is a common parent class for Raku's dual value types: ComplexStr , IntStr , NumStr , RatStr .

The dual value types (often referred to as allomorphs) allow for the representation of a value as both a string and a numeric type. Typically they will be created for you when the context is "stringy" but they can be determined to be numbers, such as in some quoting constructs :

```
my $c = <42+0i>;  say $c.^name; # OUTPUT: «ComplexStr»
my $i = <42>;     say $i.^name; # OUTPUT: «IntStr»
my $n = <42.1e0>; say $n.^name; # OUTPUT: «NumStr»
my $r = <42.1>;   say $r.^name; # OUTPUT: «RatStr»
```

As a subclass of both a Numeric class and Str , via the `Allomorph` class, an allomorph will be accepted where either is expected. However, an allomorph does not share object identity with its `Numeric` parent class- or `Str` -only variants:

```
my ($complex-str, $int-str, $num-str, $rat-str)
          = < 42+0i 42 42e10 42.1 >;

my (Complex $complex, Int $int, Num $num, Rat $rat)
          =  $complex-str, $int-str, $num-str, $rat-str;  # OK!

my Str @strings
          =  $complex-str, $int-str, $num-str, $rat-str;  # OK!

# ▫ operator cares about object identity
say 42+0i ▫ < 42+0i 42 42e10 42.1 >;  # OUTPUT: «False»
say 42    ▫ < 42+0i 42 42e10 42.1 >;  # OUTPUT: «False»
say 42e10 ▫ < 42+0i 42 42e10 42.1 >;  # OUTPUT: «False»
say 42.1  ▫ < 42+0i 42 42e10 42.1 >;  # OUTPUT: «False»
```

Please see the Numerics page for a more complete description on how to work with these allomorphs.

Methods

method ACCEPTS

```
multi method ACCEPTS(Allomorph:D: Any:D \a)
```

If the `a` parameter is Numeric (including another allomorph), checks if invocant's Numeric value ACCEPTS `a`. If the `a` parameter is Str, checks if invocant's Str value ACCEPTS `a`. If the `a` parameter is anything else, checks if both Numeric and Str values of the invocant `ACCEPTS` `a`.

```
say "5.0" ~~ <5>; # OUTPUT: «False»
say 5.0   ~~ <5>; # OUTPUT: «True»
say <5.0> ~~ <5>; # OUTPUT: «True»
```

method Bool

```
multi method Bool(::?CLASS:D:)
```

Returns `False` if the invocant is numerically `0`, otherwise returns `True`. The `Str` value of the invocant is not considered.

Note: For the `Allomorph` subclass RatStr also see Rational.Bool.

method chomp

```
method chomp(Allomorph:D:)
```

Calls Str.chomp on the invocant's `Str` value.

method chop

```
method chop(Allomorph:D: |c)
```

Calls Str.chop on the invocant's `Str` value.

method comb

```
method comb(Allomorph:D: |c)
```

Calls Str.comb on the invocant's `Str` value.

method fc

```
method fc(Allomorph:D:)
```

Calls Str.fc on the invocant's `Str` value.

method flip

```
method flip(Allomorph:D:)
```

Calls Str.flip on the invocant's `Str` value.

method lc

```
method lc(Allomorph:D:)
```

Calls Str.lc on the invocant's `Str` value.

method pred

```
method pred(Allomorph:D:)
```

Calls Numeric.pred on the invocant's numeric value.

method raku

```
multi method raku(Allomorph:D:)
```

Return a representation of the object that can be used via EVAL to reconstruct the value of the object.

method samecase

```
method samecase(Allomorph:D: |c)
```

Calls Str.samecase on the invocant's `Str` value.

method samemark

```
method samemark(Allomorph:D: |c)
```

Calls Str.samemark on the invocant's `Str` value.

method split

```
method split(Allomorph:D: |c)
```

Calls Str.split on the invocant's `Str` value.

method Str

```
method Str(Allomorph:D:)
```

Returns the `Str` value of the invocant.

method subst

```
method subst(Allomorph:D: |c)
```

Calls Str.subst on the invocant's `Str` value.

method subst-mutate

```
method subst-mutate(Allomorph:D \SELF: |c)
```

Calls Str.subst-mutate on the invocant's `Str` value.

method substr

```
method substr(Allomorph:D: |c)
```

Calls Str.substr on the invocant's `Str` value.

method substr-rw

```
method substr-rw(Allomorph:D \SELF: $start = 0, $want = Whatever)
```

Calls Str.substr-rw on the invocant's `Str` value.

method succ

```
method succ(Allomorph:D:)
```

Calls Numeric.succ on the invocant's numeric value.

method tc

```
method tc(Allomorph:D:)
```

Calls Str.tc on the invocant's `Str` value.

method tclc

```
method tclc(Allomorph:D:)
```

Calls Str.tclc on the invocant's `Str` value.

method trim

```
method trim(Allomorph:D:)
```

Calls Str.trim on the invocant's `Str` value.

method trim-leading

```
method trim-leading(Allomorph:D:)
```

Calls Str.trim-leading on the invocant's `Str` value.

method trim-trailing

```
method trim-trailing(Allomorph:D:)
```

Calls Str.trim-trailing on the invocant's `Str` value.

method uc

```
method uc(Allomorph:D:)
```

Calls Str.uc on the invocant's `Str` value.

method WHICH

```
multi method WHICH(Allomorph:D:)
```

Returns an object of type ValueObjAt which uniquely identifies the object.

```
my $f = <42.1e0>;
say $f.WHICH;     # OUTPUT: «NumStr|Num|42.1|Str|42.1e0»
```

Operators

infix cmp

```
multi sub infix:<cmp>(Allomorph:D $a, Allomorph:D $b)
```

Compare two `Allomorph` objects. The comparison is done on the `Numeric` value first and then on the `Str` value. If you want to compare in a different order then you would coerce to an `Numeric` or `Str` value first:

```
my $f = IntStr.new(42, "smaller");
my $g = IntStr.new(43, "larger");
say $f cmp $g;          # OUTPUT: «Less»
say $f.Str cmp $g.Str;  # OUTPUT: «More»
```

infix eqv

```
multi sub infix:<eqv>(Allomorph:D $a, Allomorph:D $b --> Bool:D)
```

Returns `True` if the two `Allomorph` `$a` and `$b` are of the same type, their `Numeric` values are equivalent and their `Str` values are also equivalent . Returns `False` otherwise.