Deckz

v0.1.2 2025-07-28 GPL-3.0-or-later

Render poker-style cards and full decks.

Michele Dusi
@micheledusi

DECKZ is a flexible and customizable package to render and display poker-style playing cards in Typst[1]. Use it to visualize individual cards, create stylish examples in documents, or build full decks and hands for games and illustrations.

Table of Contents

[1] https://typst.app/

Part I

How to use DECKZ

DECKZ is a Typst package designed to display playing cards in the classic poker style, using the standard French suits (hearts ♥, diamonds ♦, clubs ♣, and ♠. Whether you need to show a single card, a hand, a full deck, or a scattered heap, DECKZ provides flexible tools to visualize cards in a variety of formats and layouts. The package is ideal for games, teaching materials, or any document where clear and attractive card visuals are needed.

DECKZ offers multiple display formats, ranging from compact inline symbols to detailed, full-sized cards—so you can adapt the appearance to your specific use case. It also includes functions for visualizing groups of cards, such as hands, decks, and heaps, making it easy to represent typical card game scenarios.

This manual is organized in three parts:
1. Section I helps you get started with the main features;
2. Section II provides detailed documentation for each function, serving as a reference;
3. Section III presents practical examples that combine different features.

At the end, you'll find credits and instructions for contributing to the project.

This manual documents the most recent DECKZ release as of today: **0.1.2**.

I.1 Importing the package

To use DECKZ functionalities in your project, add this instruction to your document:

```
#import "@preview/deckz:0.1.2": deckz
```

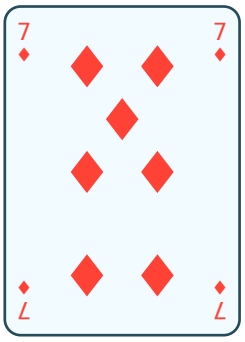
DECKZ is typically imported with the keyword `deckz`, which will be used to call the package functions.

You can then call any of the rendering functions using the `deckz` namespace.

I.2 Basic usage

The main entry point is the `#deckz.render` function:

```
#deckz.render("7D", format: "large")
```



The first argument is the **card identifier** as a string (`#deckz.render.card`). Use standard short notation like `"AH"`, `"10S"`, `"QC"`, etc., where the first letter(s) indicates the **rank**, and the last letter the **suit**.

- **Available ranks**: `A`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, `9`, `10`, `J`, `Q`, `K`.
- **Available suits**: `H` (Hearts ♥), `D` (Diamonds ♦), `C` (Clubs ♣), `S` (Spades ♠).

> Card identifier is **case-insensitive**, so `"as"` and `"AS"` are equivalent and both represent the Ace of Spades.

The second argument is optional and specifies the **format** of the card display (`#deckz.render.format`). If not provided, DECKZ functions will typically default to `medium`.

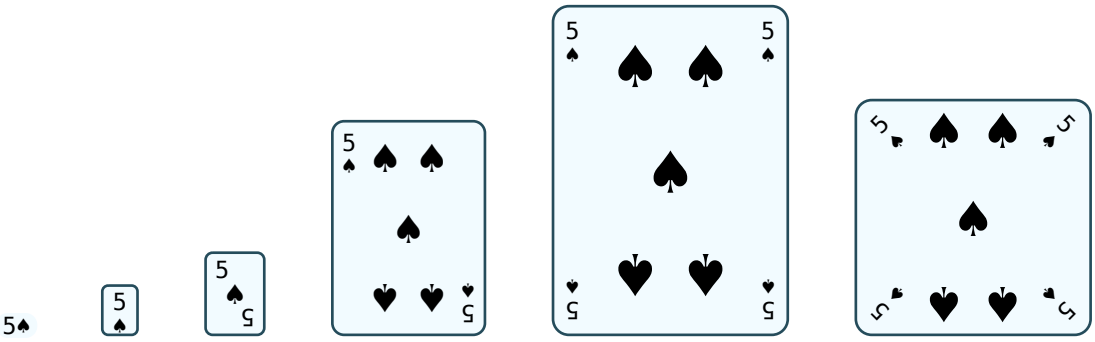
I.2.1 Formats

DECKZ provides multiple **display formats** to fit different design needs:

Format	Description
`inline`	A minimal format where the rank and suit are shown directly inline with text.
`mini`	The smallest visual format: a tiny rectangle with the rank on top and the suit at the bottom.
`small`	A compact but clear card with rank in opposite corners and the suit centered.
`medium`	A full, structured card with proper layout, two corner summaries, and realistic suit placement.
`large`	An expanded version of `medium` with corner summaries on all four sides for maximum readability.
`square`	A balanced 1:1 format with summaries in all corners and the main figure centered.

Here's an example of how the same card looks in different formats:

```
#deckz.render("5S", format: "inline") #h(1fr)
#deckz.render("5S", format: "mini") #h(1fr)
#deckz.render("5S", format: "small") #h(1fr)
#deckz.render("5S", format: "medium") #h(1fr)
#deckz.render("5S", format: "large") #h(1fr)
#deckz.render("5S", format: "square")
```

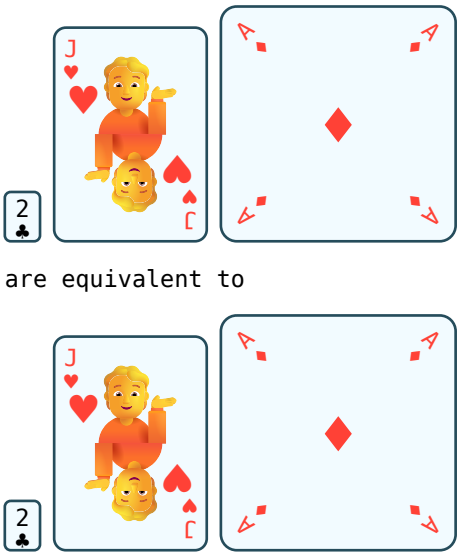


You can use any of these with the **general function** `#deckz.render`, or by calling directly the **specific format functions**:

- `#deckz.mini`,
- `#deckz.small`,
- `#deckz.medium`,
- `#deckz.large`,
- `#deckz.square`.

> All formats are **responsive to the current text size**: they scale proportionally using `em` units, making them adaptable to different layouts and styles.
>
> For reference, the summaries in larger formats (i.e. the symbols representing the rank and suit of a cards, usually placed in the card's corners) scale with the current text size, ensuring that card details remain readable even when the surrounding text is small.

```
#deckz.mini("2C")
#deckz.medium("JH")
#deckz.square("AD")

are equivalent to

#deckz.render("2C", format: "mini")
#deckz.render("JH", format: "medium")
#deckz.render("AD", format: "square")
```



If you want more examples of how to use these formats, check out Section III at the end of this document.

I.2.2 Back of Cards

To render the **back of a card**, you can use the `#deckz.back` function. This will display a generic card back design, which can be useful for games or when you want to hide the card's face.

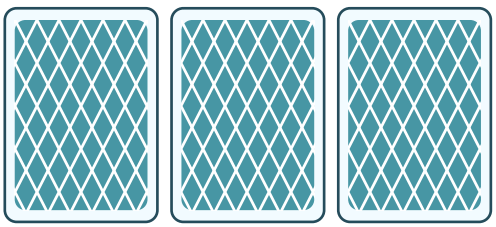
```
This is the back of a card:
#deckz.back(format: "small")
```

This is the back of a card: 

Alternatively, you can use the general `#deckz.render` function with `"back"` as the card code, which is equivalent.

> Any string other than a valid card identifier will be interpreted as a request for the back of the card (except for the empty string). The convention, however, is to use the string `"back"` for clarity.

```
// These are all equivalent:
#deckz.medium("back")
#deckz.render("back", format: "medium")
#deckz.back(format: "medium")
```



> *Coming Soon Feature.* Currently, the back of cards uses a fixed design. In future updates, DECKZ will allow you to customize the back of cards and decks.

I.3 Visualize cards together

DECKZ also provides convenient functions to render **entire decks** or **hands of cards**. Both functions produce a *CeTZ* canvas, which can be used in any context where you need to display multiple cards together.

I.3.1 Decks

The deck visualization is created with the `#deckz.deck` function, which takes a card identifier as an argument. It renders a full deck of cards, with the specified card on top.

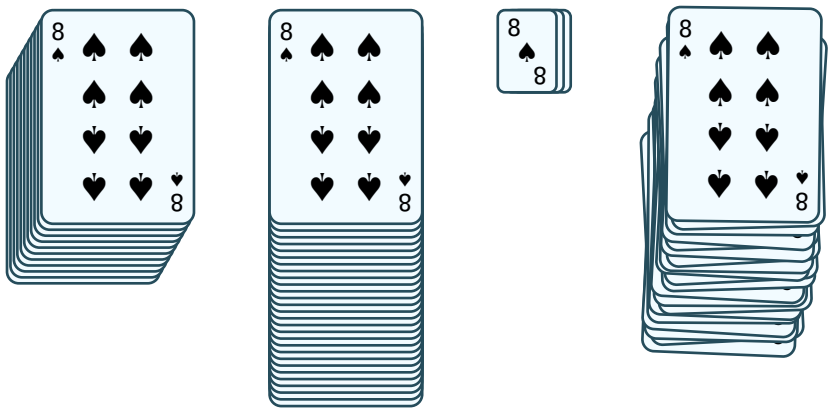
```
#deckz.deck("6D")
```

In the `#deckz.deck` function, you can also specify different parameters to **customize deck appearance**; we list here some of them.

For more information and in-depth explanations, see the documentation in Section II.

- `#deckz.deck.angle` – The direction towards which the cards are shifted.
- `#deckz.deck.height` – The height of the deck, represented as a `length`.
- `#deckz.deck.format` – The format of the cards in the deck. It can be any of the formats described above, such as `inline`, `mini`, `small`, `medium`, `large`, or `square`.

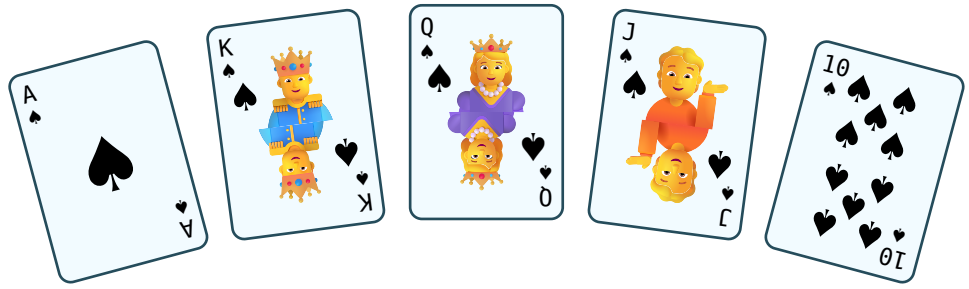
```
#stack(
  dir: ltr,
  spacing: 1cm,
  deckz.deck("8S"),
  deckz.deck("8S", angle: 90deg, height: 2.5cm),
  deckz.deck("8S", angle: 180deg, height: 8pt, format: "small"),
  deckz.deck("8S", angle: 80deg, height: 18mm, noise: 0.5)
)
```



I.3.2 Hands

The hand visualization is created with the `#deckz.hand` function, which takes a variable number of card identifiers as arguments. It renders a **hand of cards**, with the specified cards displayed side by side.

```
#deckz.hand("AS", "KS", "QS", "JS", "10S")
```



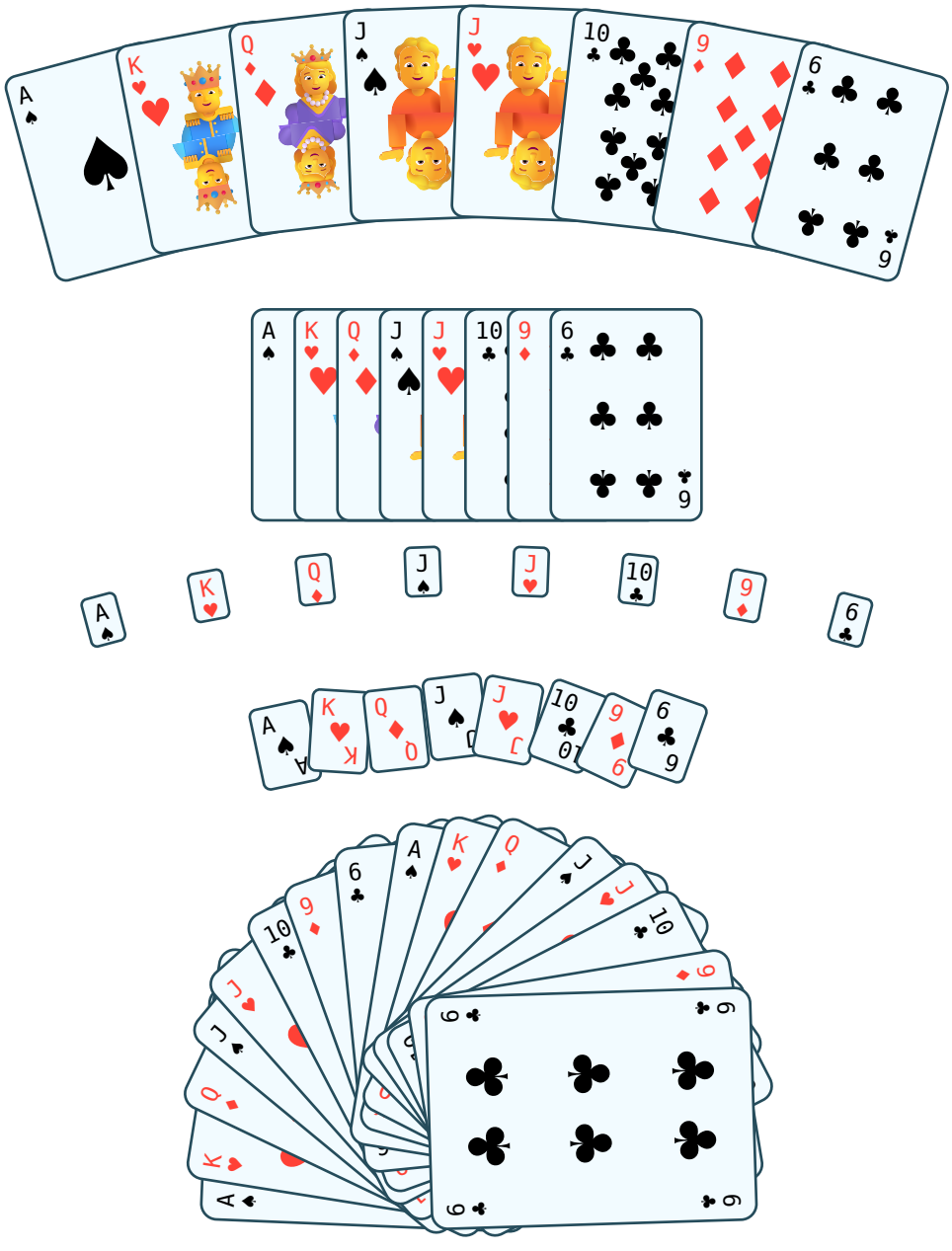
As can be seen in the example above, the cards are displayed in an arc shape, with the first card on the left and the last card on the right.

To customize such display, you can use the following parameters (more parameters for `#deckz.hand` explained in Section II):

- `#deckz.hand.angle` – The angle of the arc in degrees, i.e. the angle between the first and last cards' orientations.
- `#deckz.hand.width` – The width of the hand, i.e. the distance between the centers of the first and last card.

- `#deckz.hand.format` – The format of the cards in the deck. It can be any of the formats described above, such as `inline`, `mini`, `small`, `medium`, `large`, or `square`. The default is `medium`.

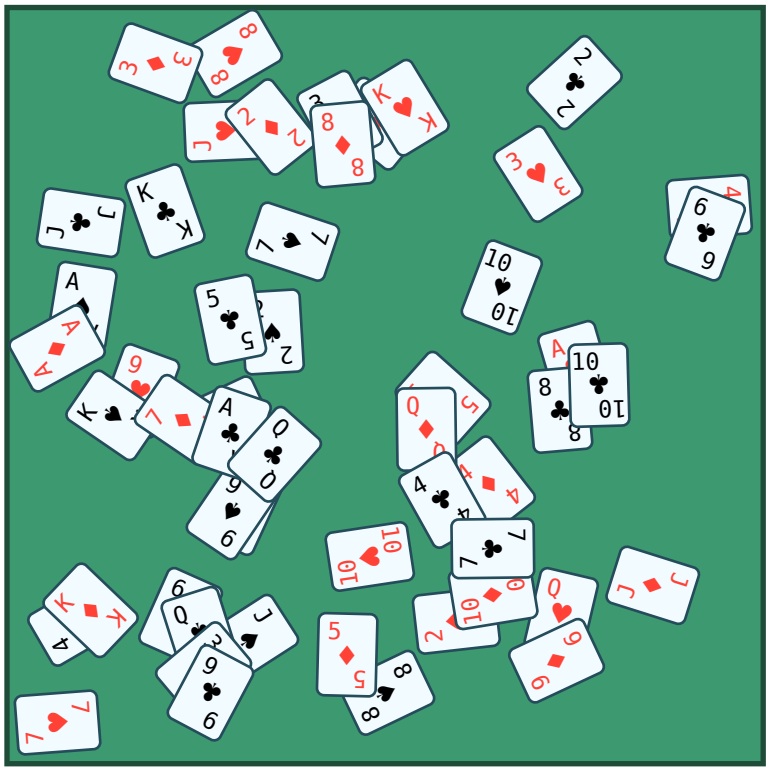
```
#let my-hand = ("AS", "KH", "QD", "JS", "JH", "10C", "9D", "6C")

#table(
  columns: (1fr),
  align: center,
  stroke: none,
  deckz.hand(..my-hand),
  deckz.hand(angle: 0deg, width: 4cm, ..my-hand),
  deckz.hand(format: "mini", ..my-hand),
  deckz.hand(width: 5cm, noise: 2, format: "small", ..my-hand),
  deckz.hand(angle: 180deg, width: 3cm, noise: 0.5, format: "large", ..(my-hand + my-hand)),
)
```



I.3.3 Heaps

DECKZ also provides a `#deckz.heap` function to display a **heap of cards**. This is similar to a hand (`#deckz.hand`), but the cards are randomly scattered within a specified area, rather than arranged in an arc. Like the hand, the heap requires a list of card identifiers as arguments, and it can be customized with various parameters.

```
#let (w, h) = (10cm, 10cm)
#box(width: w, height: h,
  fill: olive,
  stroke: olive.darken(50%) + 2pt,
)[
  #deckz.heap(format: "small", width: w, height: h, ..deckz.deck52)
]
// Note: The `deckz.deck52` array contains all 52 standard playing cards.
```



Some customization options are:

- `#deckz.heap.width` – The width of the heap, i.e. how far apart the cards will be scattered horizontally.
- `#deckz.heap.height` – The height of the heap, i.e. how far apart the cards will be scattered vertically.
- `#deckz.heap.format` – The format of the cards in the heap. It can be any of the formats described above, such as `inline`, `mini`, `small`, `medium`, `large`, or `square`. The default is `medium`.

See also Section II for the full list of parameters of `#deckz.heap`.

I.4 Card customization

DECKZ allows for some **customization of the card appearance**, such as colors and styles.

> *Coming Soon Feature.* This functionality is not fully supported yet: please, come back for the next releases.

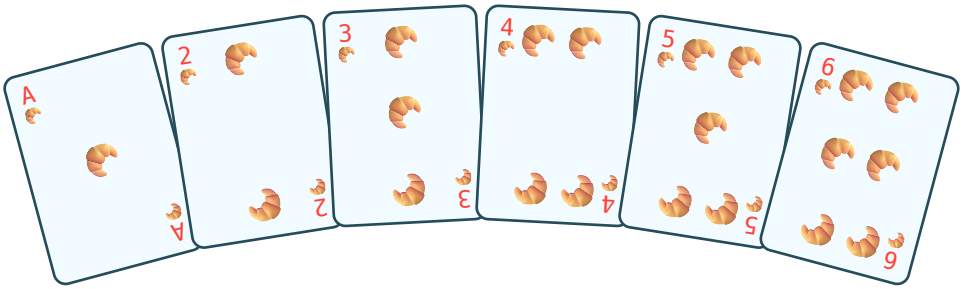
I.4.1 Custom Suits

DECKZ will also allow you to define custom suits, so you can use your own symbols or images instead of the standard hearts, diamonds, clubs, and spades.

Even though this feature is not yet implemented, you can still use custom suits by defining your own `show` rule for the emoji suits. In fact, DECKZ uses the `emoji.suit.*` symbols to render the standard suits, so you can override them with your own definitions.

For example, if you want to use a *croissant emoji* 🥐 as a custom suit instead of *diamonds* ♦, you can define it like this:

```
#show emoji.suit.diamond: text(size: 0.7em, emoji.croissant)
#deckz.hand(..deckz.deck52.slice(26, 32))
```



> The **resizing** of the emoji in the previous example is used to make it fit better in the card layout. When you're defining your own `show` rule for suits customization, you may need to adjust their size as needed.

Part II
Documentation

II.1 Card Visualization

This section provides a comprehensive overview of the DECKZ package's **card visualization** capabilities. It presents the available formats and how to use them effectively.

```
#deckz.inline                #deckz.mini                #deckz.square
#deckz.large                 #deckz.render
#deckz.medium                #deckz.small
```

#deckz.inline(card) → content

Renders a card with the "**inline**" format. The card is displayed in a compact style: text size is coherent with the surrounding text, and the card is rendered as a simple text representation of its rank and suit.

```
#lorem(10)
#deckz.inline("AS"), #deckz.inline("3S")
#lorem(10)
#deckz.inline("KH").
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do. A♠, 3♠ Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do. K♥.

Argument

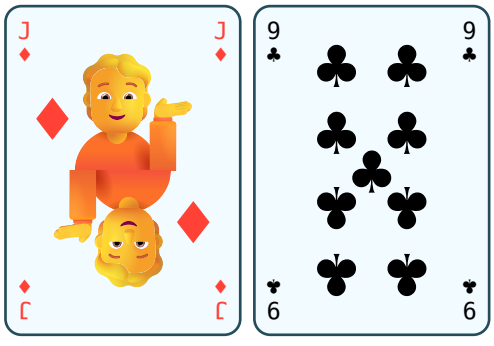
card — str

The code of the card you want to represent.

#deckz.large(card) → content

Renders a card with the "**large**" format, emphasizing the card's details: all four corners are used to display the rank and suit, with a large central representation. Like other formats, the large format is responsive to text size; corner summaries are scaled accordingly to the current text size.

```
#deckz.large("JD")
#deckz.large("9C")
```



Argument

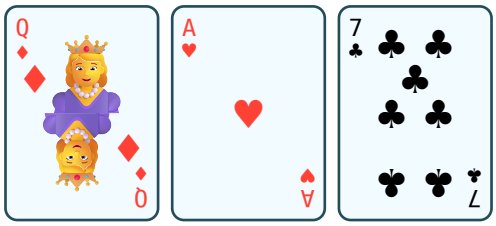
⟨card⟩ `str`

The code of the card you want to represent.

#deckz.medium(⟨card⟩) ⟶ content

Renders a card with the "**medium**" format: a full, structured card layout with two corner summaries and realistic suit placement. The medium format is usually the default format for card rendering in DECKZ.

```
#deckz.medium("QD")
#deckz.medium("AH")
#deckz.medium("7C")
```



Argument

⟨card⟩ `str`

The code of the card you want to represent.

#deckz.mini(⟨card⟩) ⟶ content

Renders a card with the "**mini**" format. The card is displayed in a very compact style, suitable for dense layouts. The frame size is responsive to text, and it contains a small representation of the card's rank and suit.

```
#deckz.mini("JC")
#deckz.mini("AH")
#deckz.mini("5S")
#deckz.mini("9D")
#deckz.mini("4H")
#deckz.mini("3C")
#deckz.mini("2D")
#deckz.mini("KS")
```

Argument

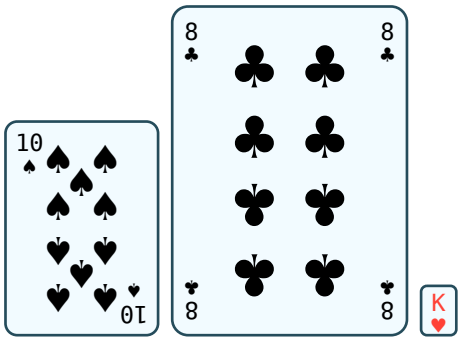
⟨card⟩ `str`

The code of the card you want to represent.

#deckz.render(⟨card⟩, ⟨format⟩: "medium", ⟨noise⟩: none) ⟶ content

Render function to view cards in different formats. This function allows you to specify the format of the card to be rendered. Available formats include: inline, mini, small, medium, large, and square.

```
#deckz.render("10S")
#deckz.render(format: "large", "8C")
#deckz.render(format: "mini", "KH")
```



Argument

(card) str

The code of the card you want to represent.

Argument

(format): "medium" str

The selected format (inline, mini, small, medium, large, and square). Default value is "medium".

Argument

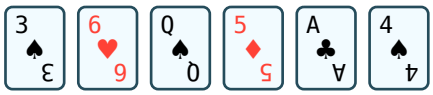
(noise): none float | none

The amount of "randomness" in the placement and rotation of the card. Default value is "none" or "0", which corresponds to no variations. A value of 1 corresponds to a "standard" amount of noise, according to Deckz style. Higher values might produce crazy results, handle with care.

#deckz.small((card)) → content

Renders a card with the "**small**" format. The card is displayed in a concise style, balancing readability and space: the card's rank is shown symmetrically in two corners, with the suit displayed in the center.

```
#deckz.small("3S")
#deckz.small("6H")
#deckz.small("QS")
#deckz.small("5D")
#deckz.small("AC")
#deckz.small("4S")
```



Argument

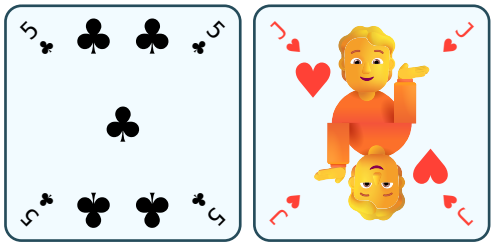
(card) str

The code of the card you want to represent.

#deckz.square((card)) → content

Renders a card with the "**square**" format, i.e. with a frame layout with 1:1 ratio. This may be useful for grid layouts or for situations where the cards are often rotated in many directions, because the corner summaries are placed diagonally.

```
#deckz.square("5C")
#deckz.square("JH")
```



Argument

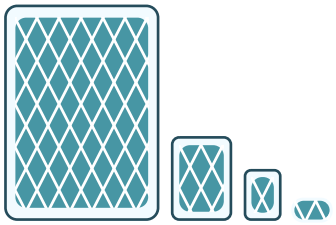
⟨card⟩ `str`

The code of the card you want to represent.

#deckz.back(⟨format⟩: "medium") ⟶ **content**

Renders the back of a card in the specified format. Currently, it only supports one style, which is a simple rectangle with a rhombus pattern.

```
#deckz.back(format: "medium")
#deckz.back(format: "small")
#deckz.back(format: "mini")
#deckz.back(format: "inline")
```



Argument

⟨format⟩: "medium"

The format of the card back, defaults to `medium`. Available formats: `inline`, `mini`, `small`, `medium`, `large`, `square`.

II.2 Group Visualization

This section covers the **group visualization** features of the DECKZ package, i.e. all functions that allow you to visualize groups of cards, such as hands, decks, and heaps.

(More functions and options will be added in the future).

`#deckz.deck` `#deckz.hand` `#deckz.heap`

```
#deckz.deck(
  (angle): 60deg,
  (height): 1cm,
  (noise): none,
  (format): "medium",
  (top-card)
) ⟶ content
```

Renders a **stack** of cards, as if they were placed one ontop of each other. Calculates the number of cards based on the given height (`#deckz.deck.height`), and spaces them evenly along the specified angle (`#deckz.deck.angle`). Each card is rendered with a positional shift to create a fanned deck appearance.

```
#deckz.deck(
  angle: 20deg,
  height: 1.5cm,
  "7D"
)
```

Argument

`(angle): 60deg` `angle`

The **angle** at which the deck is fanned out. Default is `60deg`.

Argument

`(height): 1cm` `height`

The total **height** of the deck stack. This determines how many cards are rendered in the stack, as one card is displayed for every `2.5pt` of height.

Argument

`(noise): none` `float` | `none`

The amount of "**randomness**" in the placement and rotation of the card. Default value is "none" or "0", which corresponds to no variations. A value of 1 corresponds to a "standard" amount of noise, according to Deckz style. Higher values might produce crazy results, handle with care.

Argument

⟨format⟩: "medium" `str`

The **format** to use for rendering each card. Default is “medium”.

Argument

⟨top-card⟩ `str`

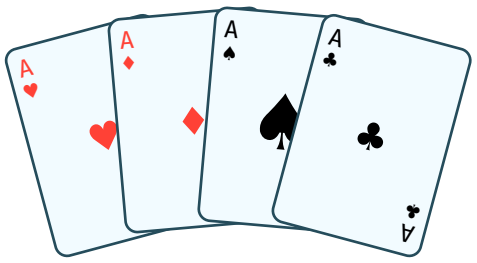
The **top card** in the deck, with standard code representation.

```
#deckz.hand(
  ⟨angle⟩: 30deg,
  ⟨width⟩: 10cm,
  ⟨noise⟩: none,
  ⟨format⟩: "medium",
  ..⟨cards⟩
) ⟶ content
```

Displays a **sequence of cards** in a horizontal hand layout. Optionally applies a slight rotation to each card, creating an arched effect.

This function is useful for displaying a hand of cards in a visually appealing way. It accepts any number of cards, each represented by a string identifier (e.g., `"AH"` for Ace of Hearts).

```
#deckz.hand(
  width: 100pt,
  "AH", "AD", "AS", "AC"
  // Poker of Aces
)
```



Argument

⟨angle⟩: 30deg `angle`

The **angle** between the first and last card, i.e. the angle covered by the arc.

Argument

⟨width⟩: 10cm `length`

The **width** of the hand, i.e. the distance between the first and last card.

Argument

⟨noise⟩: none `float` | `none`

The amount of “**randomness**” in the placement and rotation of the card. Default value is “none” or “0”, which corresponds to no variations. A value of “1” corresponds to a “standard” amount of noise, according to DECKZ style. Higher values might produce crazy results, handle with care.

Argument

`(format): "medium"` `str`

The **format** of the cards to render. Default is “medium”. Available formats: `inline`, `mini`, `small`, `medium`, `large`, `square`.

Argument

`..(cards)` `array`

The list of **cards** to display, with standard code representation.

```
#deckz.heap(
  (format): "medium",
  (width): 10cm,
  (height): 10cm,
  (exceed): false,
  ..(cards)
) → content
```

Renders a **heap** of cards, randomly placed in the given area. The cards are placed in a random position within the specified width (`#deckz.heap.width`) and height (`#deckz.heap.height`), with a random rotation applied to each card. The `#deckz.heap.exceed` parameter controls whether cards can exceed the specified frame dimensions or not.

```
#deckz.heap(
  format: "small",
  width: 5cm,
  height: 4cm,
  "7D", "8D", "9D", "10D", "JD"
)
```

Argument

`(format): "medium"` `str`

The **format** to use for rendering each card. Default is “medium”.

Argument

`(width): 10cm` `length`

The **horizontal dimension** of the area in which cards are placed.

Argument

`(height): 10cm` `length`

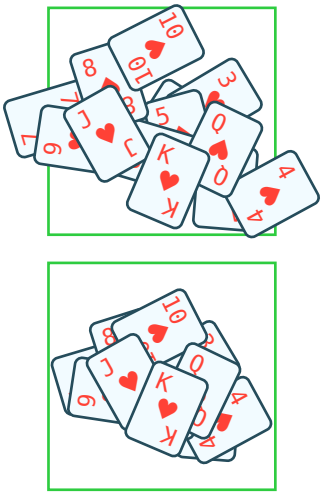
The **vertical dimension** of the area in which cards are placed.

Argument

`(exceed): false` `bool`

If `true`, allows cards to **exceed the frame** with the given dimensions. When the parameter is `false`, instead, cards placement considers a margin of half the card length on all four sides. This way, it is guaranteed that cards are placed within the specified frame size. Default is `false`.

```
// Example with `exceed: true`
#box(width: 3cm, height: 3cm, stroke:
green)[
  #place(center + horizon, deckz.heap(
    format: "small",
    width: 3cm,
    height: 3cm,
    exceed: true,
    ..deckz.deck52.slice(0, 13)
  ))
]

// Example with `exceed: false`
#box(width: 3cm, height: 3cm, stroke:
green)[
  #place(center + horizon, deckz.heap(
    format: "small",
    width: 3cm,
    height: 3cm,
    exceed: false,
    ..deckz.deck52.slice(0, 13)
  ))
]
```



Argument

`..(cards)` `array`

The **cards to display**, with standard code representation. The last cards are represented on top of the previous one, as the rendering follows the given order.

II.3 Data

This section provides an overview of the data structures used in the DECKZ package, including suits, ranks, and cards. It explains how these data structures are organized and how to access them.

#suits `dictionary`

A mapping of all **suit symbols** utilized in DECKZ.

```
#deckz.suits
```

```
(
  heart: symbol("♥"),
  spade: symbol("♠"),
  diamond: symbol("♦"),
  club: symbol("♣"),
)
```

Primarily intended for internal use within higher-level functions, but can also be accessed directly, for example, to iterate over the four suits.

```
#stack(
  dir: ltr,
  spacing: 1fr,
  ..deckz.suits.values()
)
```

♥ ♠ ♦ ♣

#ranks `dictionary`

A mapping of all **rank symbols** utilized in DECKZ.

```
#deckz.ranks
```

```
(
  ace: "A",
  two: "2",
  three: "3",
  four: "4",
  five: "5",
  six: "6",
  seven: "7",
  eight: "8",
  nine: "9",
  ten: "10",
  jack: "J",
  queen: "Q",
  king: "K",
)
```

This dictionary is primarily intended for internal use within higher-level functions, but can also be accessed directly, for example, to iterate over the ranks.

```
#table(
  columns: 5 * (1fr, ),
  ..deckz.ranks.keys()
)
```

ace	two	three	four	five
six	seven	eight	nine	ten
jack	queen	king		

#cards52 `dictionary`

This is a dictionary of **all the cards in a deck**.

It is structured as `cards.<suit-k>.<rank-k>`, where:
- `<suit-k>` is one of the keys from the `suits` dictionary, and
- `<rank-k>` is one of the keys from the `ranks` dictionary.

The value associated with each (rank, suit) pair is the **card code**, which is a string in the format `<rank-s><suit-s>`, where `<rank-s>` is the rank symbol and `<suit-s>` is the first letter of the suit key in uppercase.

```
#deckz.cards52.heart.ace // Returns "AH"
#deckz.cards52.spade.king // Returns "KS"
#deckz.cards52.diamond.ten // Returns "10D"
#deckz.cards52.club.three // Returns "3C"
```

```
AH KS 10D 3C
```

This dictionary can be used to access any card in a standard deck of 52 playing cards by its suit and rank, and use it in various functions that require a card code.

Here is an example using the `#deckz.hand` function:

```
#deckz.hand(
  format: "small",
  width: 128pt,
  angle: 90deg,
  ..deckz.cards52.heart.values(),
)
```

#deck52 `array`

A list of all the cards in a standard deck of 52 playing cards. It is a *flat* list of **card codes**, where each code is a string in the format `<rank-s><suit-s>`, where `<rank-s>` is the rank symbol and `<suit-s>` is the first letter of the suit key in uppercase. It is created programmatically from the `suits` and `ranks` dictionaries.

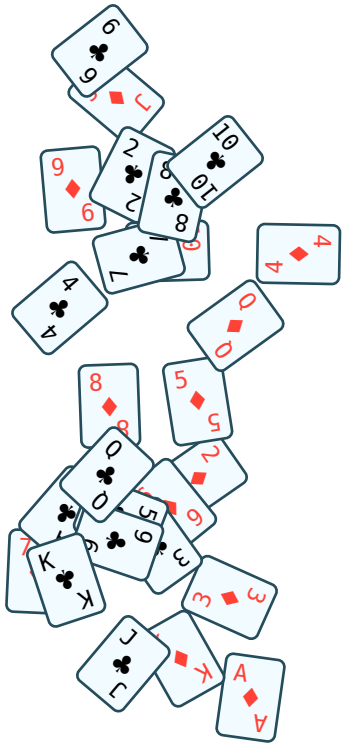
```
#table(
  columns: 13,
  align: center,
  stroke: none,
 ..deckz.deck52,
)
```

AH	2H	3H	4H	5H	6H	7H	8H	9H	10H	JH	QH	KH
AS	2S	3S	4S	5S	6S	7S	8S	9S	10S	JS	QS	KS
AD	2D	3D	4D	5D	6D	7D	8D	9D	10D	JD	QD	KD
AC	2C	3C	4C	5C	6C	7C	8C	9C	10C	JC	QC	KC

This array can be used in various functions that require a list of card codes, such as `#deckz.hand`, `#deckz.deck`, or `#deckz.heap`.

Here is an example using the `#deckz.heap` function:

```
#deckz.heap(
  format: "small",
  width: 128pt,
  height: 10cm,
  ..deckz.deck52.slice(26, 52),
)
```



II.4 Language-aware Card Symbols

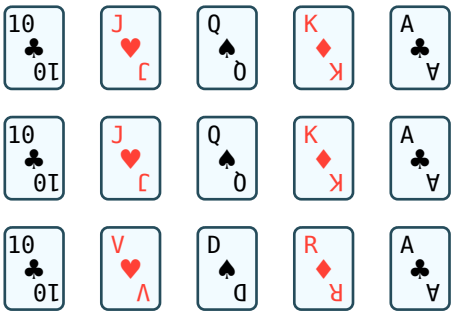
DECKZ automatically adapts the rendering of card rank symbols based on the document's language. This process is seamless: users only need to set the desired language using the `text` command, and DECKZ will adjust the symbols accordingly. No additional configuration is required.

This feature is powered by the linguify[2] package.

Currently supported languages and their rank symbols:

- **English**: `A`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, `9`, `10`, `J`, `Q`, `K`
- **Italian**: `A`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, `9`, `10`, `J`, `Q`, `K`
- **French**: `A`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, `9`, `10`, `V`, `D`, `R`

```
#let seq = ("10C", "JH", "QS", "KD", "AC")

#set text(lang: "en")
#stack(dir: ltr, spacing:
5mm, ..seq.map(deckz.small))

#set text(lang: "it")
#stack(dir: ltr, spacing:
5mm, ..seq.map(deckz.small))

#set text(lang: "fr")
#stack(dir: ltr, spacing:
5mm, ..seq.map(deckz.small))
```



[2] https://typst.app/universe/package/linguify

Part III

Examples

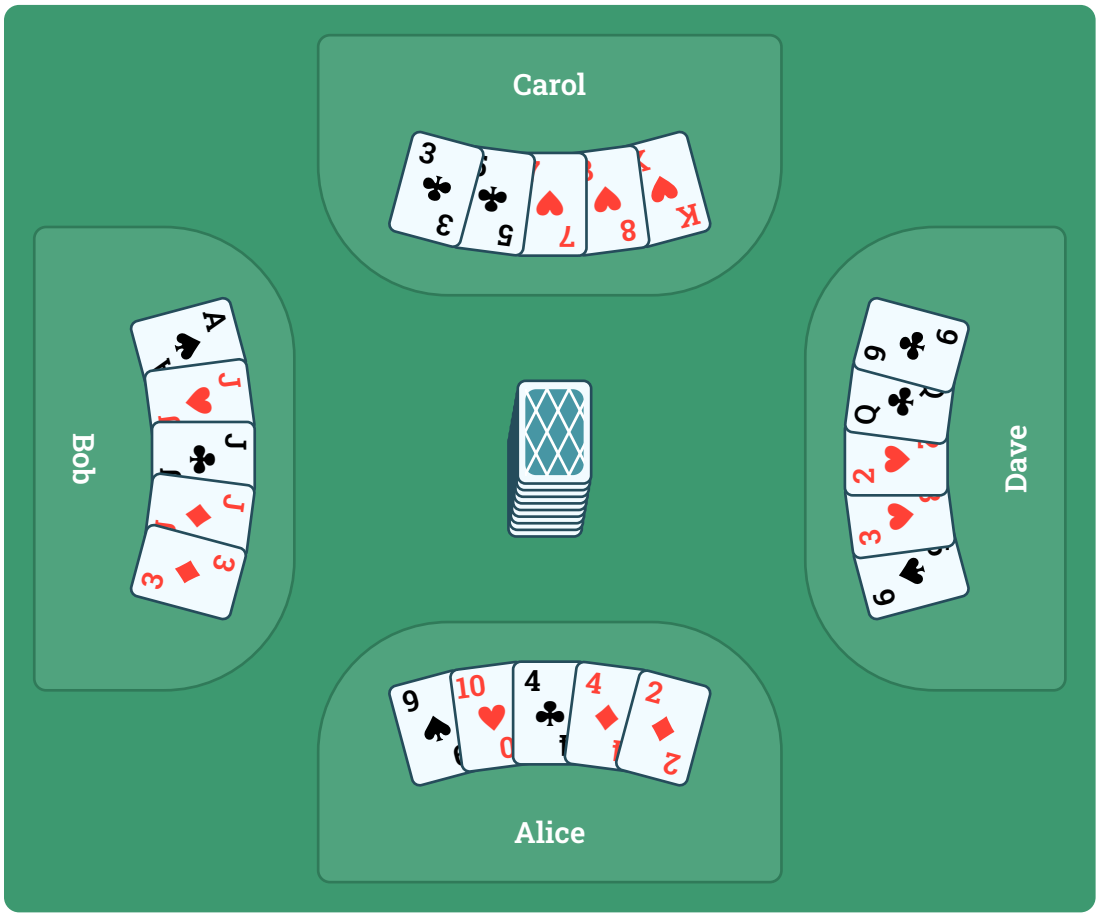
The following examples showcase more advanced and interesting use cases of DECKZ. In this Section, you can find how DECKZ can be used to represent real game states, compare card formats, and display entire decks in creative ways.

III.1 Displaying the current state of a game

You can use DECKZ to display the **current state of a game**, such as the cards in hand, the cards on the table, and the deck.

```
#let player-mat(body) = box(
  stroke: olive.darken(20%),
  fill: olive.lighten(10%),
  radius: (top: 50%, bottom: 5%),
  inset: 15%,
  body
)

#text(white, font: "Roboto Slab", weight: "semibold")[
  #box(fill: olive,
    width: 100%, height: 12cm,
    inset: 4mm, radius: 2mm
  )[

    #place(center + bottom)[
      #player-mat[
       #deckz.hand(format: "small", width: 3cm, "9S", "10H", "4C", "4D", "2D")
        Alice
      ]
    ]
    #place(left + horizon)[
      #rotate(90deg, reflow: true)[
        #player-mat[
         #deckz.hand(format: "small", width: 3cm, "AS", "JH", "JC", "JD", "3D")
          #align(center)[Bob]
        ]
      ]
    ]
    #place(center + top)[
      #rotate(180deg, reflow: true)[
        #player-mat[
         #deckz.hand(format: "small", width: 3cm, "KH", "8H", "7H", "5C", "3C")
          #rotate(180deg)[Carol]
        ]
      ]
```

```
    ]
    #place(right + horizon)[
      #rotate(-90deg, reflow: true)[
        #player-mat[
         #deckz.hand(format: "small", width: 3cm, "6S", "3H", "2H", "QC", "9C")
          #align(center)[Dave]
        ]
      ]
    ]
    #place(center + horizon)[
      #deckz.deck(format: "small", angle: 80deg, height: 8mm, "back")
    ]
  ]
]

In this situation, Alice has a *Pair of Four* (#deckz.inline("4C"),
#deckz.inline("4D")). _What should the player do?_
```



In this situation, Alice has a **Pair of Four** (4♣, 4♦). *What should the player do?*

III.2 Comparing different formats

You can use DECKZ to **compare different formats** of the same card, or to show how a card looks in different contexts.

```
#set table(stroke: 1pt + white, fill: white)
#set text(font: "Arvo", size: 0.8em)
#block(fill: gray.lighten(60%), inset: 10pt)[

  #let example-cards = ("AS", "5H", "10C", "QD")

   #text(blue)[`inline`] --- A minimal format where the rank and suit are displayed directly within the flow of text; perfect for quick references.
  #table(align: center, columns: (1fr,) * 4,
    ..example-cards.map(deckz.inline),
  )

   #text(blue)[`mini`] --- The smallest visual format: a compact rectangle showing the rank at the top and the suit at the bottom.
  #table(align: center, columns: (1fr,) * 4,
    ..example-cards.map(deckz.mini),
  )

   #text(blue)[`small`] --- A slightly larger card with rank indicators on opposite corners and a central suit symbol; ideal for tight layouts with better readability.
  #table(align: center, columns: (1fr,) * 4,
    ..example-cards.map(deckz.small),
  )

 #text(blue)[`medium`] --- A fully structured card layout featuring proper suit placement and figures. Rank and suit appear in two opposite corners, offering a realistic visual.
  #table(align: center, columns: (1fr,) * 4,
    ..example-cards.map(deckz.medium),
  )

 #text(blue)[`large`] --- The most detailed format, with corner summaries on all four corners and an expanded layout; great for presentations or printable decks.
  #table(align: center, columns: (1fr,) * 4,
    ..example-cards.map(deckz.large),
  )
]
```

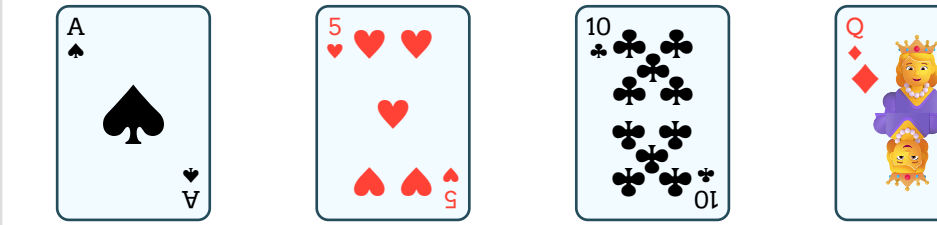
`inline` — A minimal format where the rank and suit are displayed directly within the flow of text; perfect for quick references.

A♠	5♥	10♣	Q♦

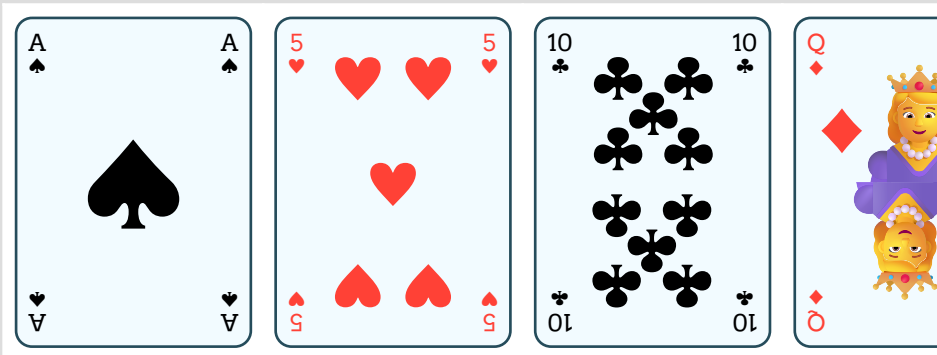
`mini` — The smallest visual format: a compact rectangle showing the rank at the top and the suit at the bottom.

`small` — A slightly larger card with rank indicators on opposite corners and a central suit symbol; ideal for tight layouts with better readability.

`medium` — A fully structured card layout featuring proper suit placement and figures. Rank and suit appear in two opposite corners, offering a realistic visual.



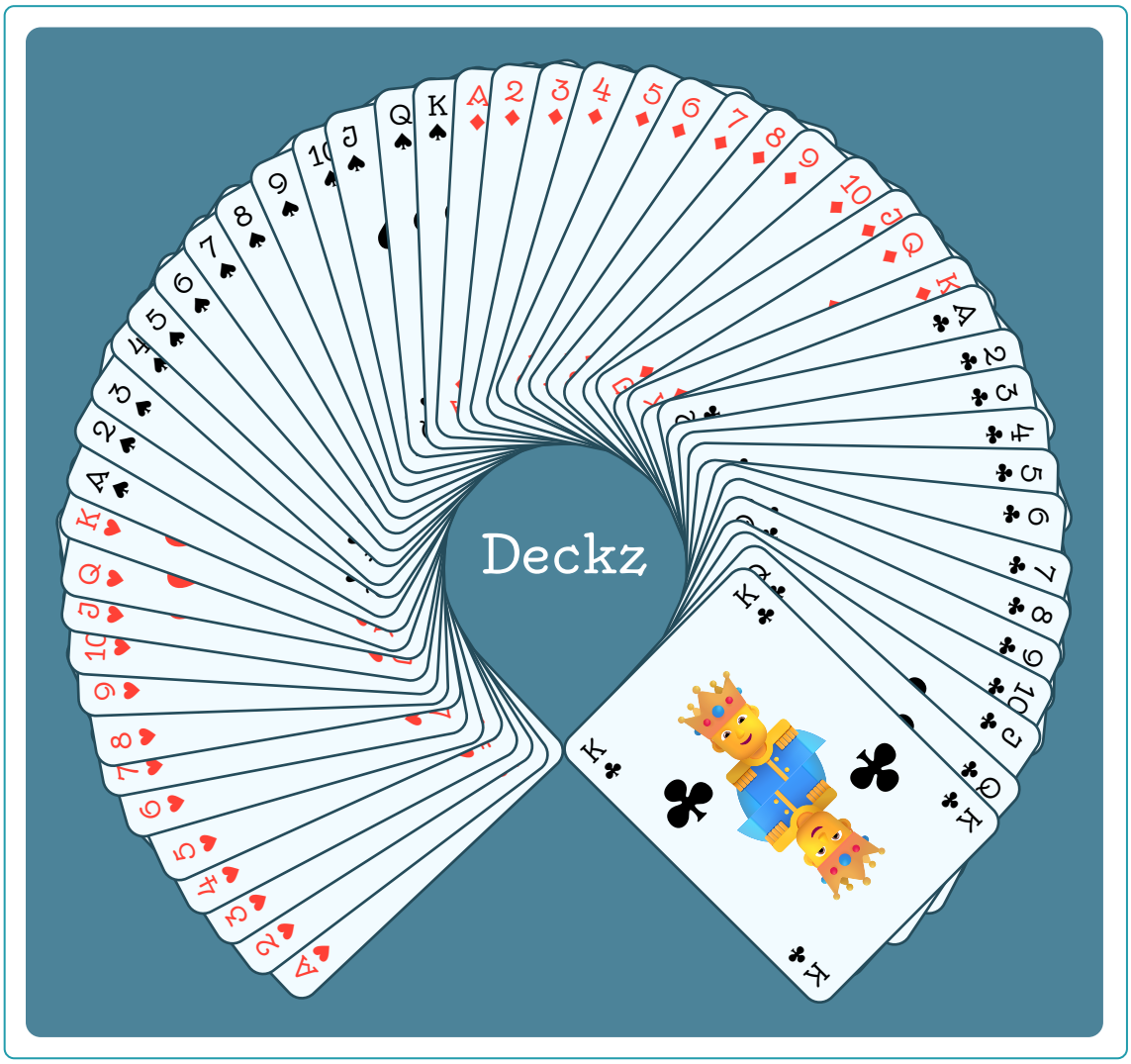
`large` — The most detailed format, with corner summaries on all four corners and an expanded layout; great for presentations or printable decks.



III.3 Displaying a full deck

You can use DECKZ to display a **full deck of cards**, simply by retrieving the `deckz.deck52` array, which contains all 52 standard playing cards.

```
#text(white, font: "Oldenburg", size: 10pt)[

  #block(fill: aqua.darken(40%),
    inset: 4mm,
    radius: 2mm,
  )[

    #deckz.hand(
      angle: 270deg,
      width: 5.8cm,
      format: "large",
      noise: 0.35,
      ..deckz.deck52
    )

    #place(center + horizon)[
      #text(size: 20pt, baseline: 8pt)[
        Deckz
      ]
    ]
  ]
]
```


Deckz

Part IV
Credits

This package is created by **Michele Dusi**[3] and is licensed under the GNU General Public License v3.0[4].

The **name** is inspired by Typst's drawing package CeTZ[5]: it mirrors its sound while hinting at its own purpose: rendering card decks.

All **fonts** used in this package are licensed under the SIL Open Font License, Version 1.1[6] (*Oldenburg*[7], *Arvo*[8]) or the Apache License, Version 2.0[9] (*Roboto Slab*[10]).

The **card designs** are inspired by the standard playing cards, with suit symbols taken from the emoji library of Typst[11].

This project owes a lot to the creators of these **Typst packages**, whose work made DECKZ possible:

- CeTZ[12], for handling graphics and for the name inspiration.
- Suiji[13], for random number generation.
- Mantys[14], Tidy[15], and Codly[16], for documentation.
- Octique[17] and Showybox[18], for documentation styling.

Special thanks to everyone involved in the development of the Typst[19] language and engine, whose efforts made the entire ecosystem possible.

IV.1 Contributing

Found a bug, have an idea, or want to contribute? Feel free to open an **issue** or **pull request** on the GitHub repository[20].

Made something cool with Deckz? Let me know — I'd love to feature your work!

---

[3] https://github.com/micheledusi
[4] https://www.gnu.org/licenses/gpl-3.0.en.html
[5] https://typst.app/universe/package/cetz
[6] https://openfontlicense.org
[7] https://fonts.google.com/specimen/Oldenburg
[8] https://fonts.google.com/specimen/Arvo
[9] http://www.apache.org/licenses/
[10] https://fonts.google.com/specimen/Roboto+Slab
[11] https://typst.app/docs/img/reference/symbols/emoji/
[12] https://typst.app/universe/package/cetz
[13] https://typst.app/universe/package/suiji
[14] https://github.com/jneug/typst-mantys
[15] https://github.com/Mc-Zen/tidy
[16] https://typst.app/universe/package/codly
[17] https://typst.app/universe/package/octique/
[18] https://typst.app/universe/package/showybox/2.0.4
[19] https://typst.app/about/
[20] https://github.com/micheledusi/Deckz

Part V

Index