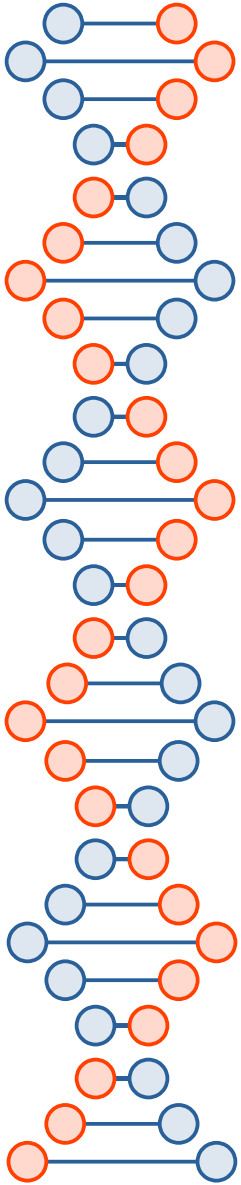
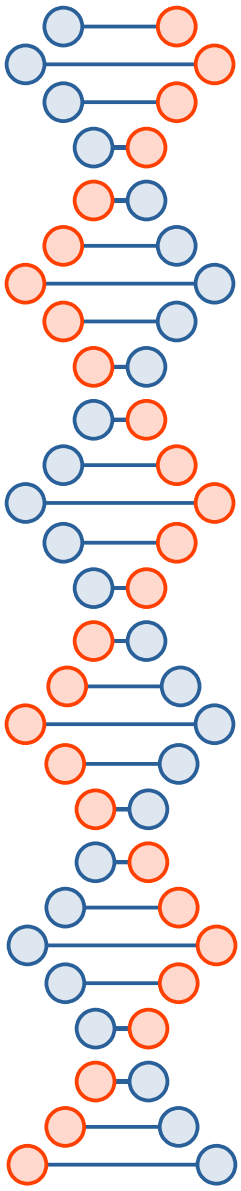
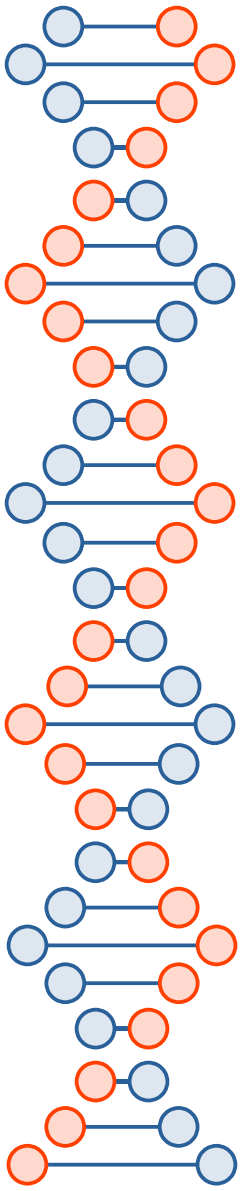


Describing Environment Variables in Worlds using `*` imports



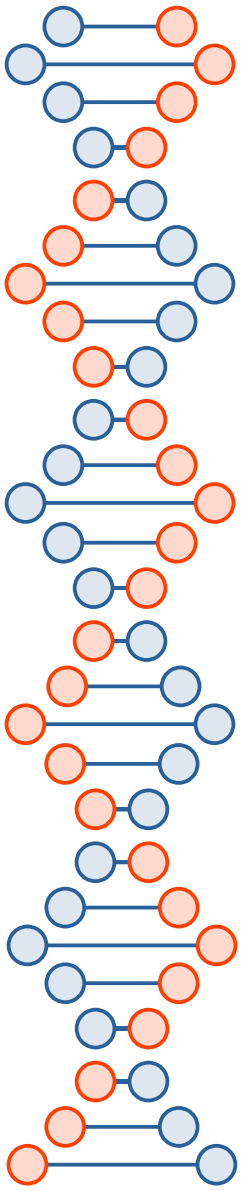
wasi:cli/Command

```
world SyncCommand {
  import filesystem: "wasi:filesystem/types"
  import socket: "wasi:sockets/types"
  import clock: "wasi:clocks/types"
  import random: "wasi:random/csprng"
  import poll: "wasi:poll/oneoff"
  import stdout: "wasi:logging/stdout"
  import stderr: "wasi:logging/stderr”

  export cli: "wasi:cli/main"
}
```



wasi:cli/main

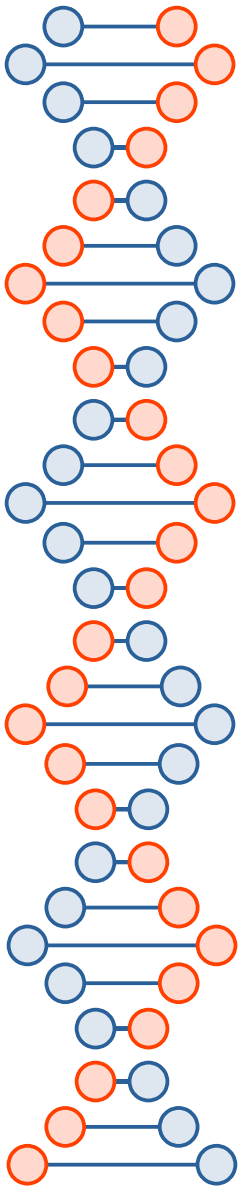
```
use { descriptor } from wasi_filesystem
use { socket } from wasi_sockets


variant preopen {
    descriptor(descriptor),
    socket(socket),
}


main: func(
    args: list<string>,
    stdin: file,
    preopens: list<tuple<string, preopen>>,
) -> result<_, _>
```



Additional handles: link-time capabilities

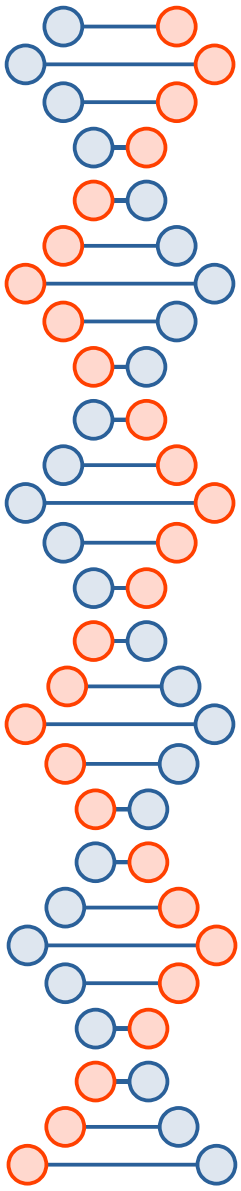
- For example, wasi:clocks/default

```
use { monotonic-clock, wall-clock } from wasi_clocks


default-monotonic-clock: func() -> monotonic-clock

default-wall-clock: func() -> wall-clock
```



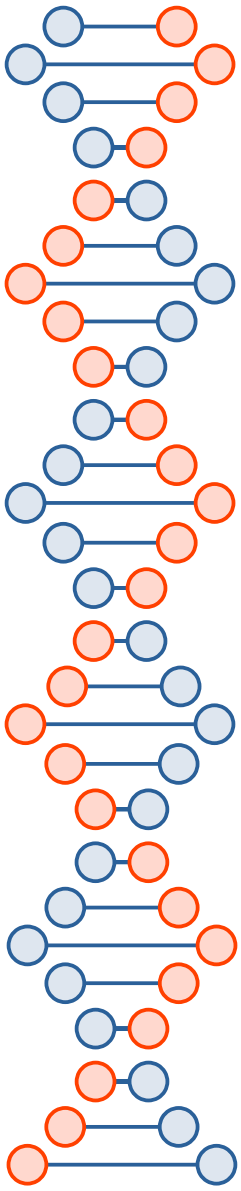
Environment variable considerations for WASI

- Portability
  - Case sensitivity
  - Platform-specific limits
  - Naming conventions for platform variables
- Sandboxing
  - Usernames
  - Host filesystem paths, which may be remapped within the guest
  - Authentication tokens in environment variables
- Composition
  - If two programs communicate, do they share an environment?



Some approaches

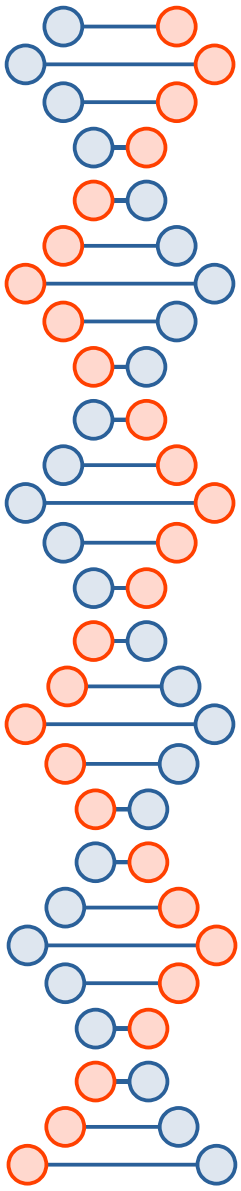
- Run wasm engines inside a separate sandbox?
  - Some users may do this, but not all
- Engines automatically deny-list known sensitive variables?
  - It’s hard to automatically filter everything that might be a problem, or without breaking actual use cases.
- Give end users control?
  - End users don’t want to be typing --env all the time
  - Need some form of global or per-program config?
- What if WASI programs came with a manifest that said which environment variables they use?
  - Most getenv calls are passed string constants
  - Make programs more self-describing and composable



Env vars as imports in Wit?

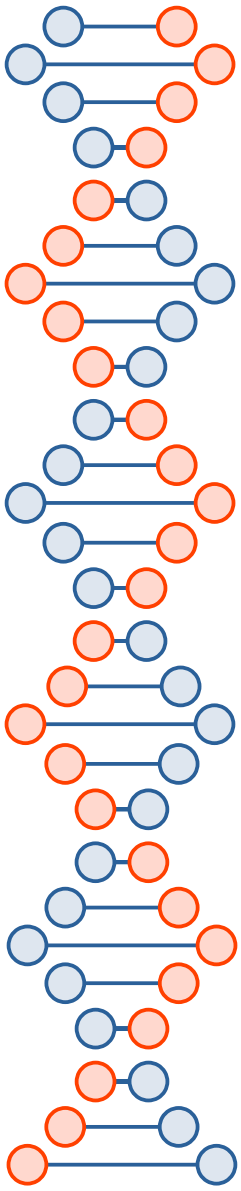
```
lang: func() -> string
tz: func() -> string
my-program-custom-setting: func() -> string
my-program-another-custom-setting: func() -> string
```

What is the type of a component using this?



Wits and Worlds

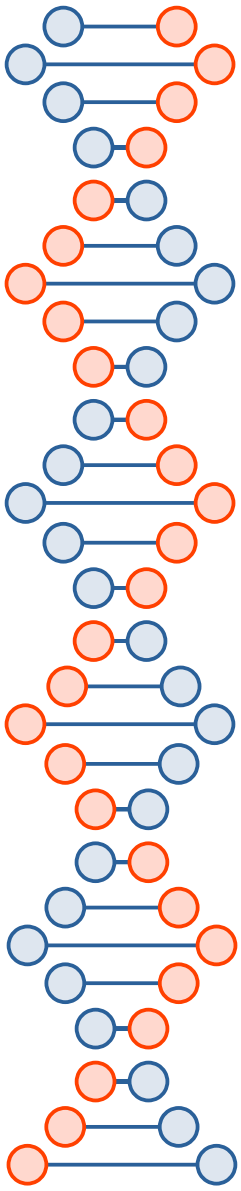
- A Wit describes an interface
  - `foo: func(x: float32) → string`
  - Any Wit be used in either direction, import or export
- A World collects Wits with directions
  - `import format: wasi:format/floats`
  - A World is a component type?
- But with plain imports, each set of environment variables would imply a different component type



Star imports in Wit

```
*: func() -> string
```

A World is a family of component types



More uses for star imports in Wit

- Define worlds with namespaces of resources, resolved at link time
  - Http backends
  - Kv stores
  - Config values
  - Message queues
  - Pubsub providers