Sway Basics Workshop Cheat Sheet

Contracts

```
contract;

abi SomeAbi {
    . . .
}

storage {
    . . .
}

impl SomeAbi for Contract {
    . . .
}
```

Abis

```
abi SomeAbi {
    #[storage(read, write)]
    fn do_something(x: (u8, b256)) -> [u256; 5];

    #[storage(read)]
    fn count() -> u64;

    fn no_storage_access();
}
```

Contract Impl

```
impl SomeAbi for Contract {
    #[storage(read, write)]
    fn do_something(x: (u8, b256)) -> [u256; 5] {
        ...
    }

    #[storage(read)]
    fn count() -> u64 {
        ...
    }
    ...
}
```

Storage

```
use std::storage::storage_vec::*;

storage {
    val: u64 = 0,
    vec: StorageVec<u64> = StorageVec{},
}

fn some_contract_method(input: u64) {
    let v = storage.val.read();
    storage.val.write(input);
    let v = storage.vec.get(0).unwrap_or(42);
    storage.vec.push(input);
}
```

Structs

```
struct Foo {
    bar: u64,
    baz: bool,
}

impl Foo {
    fn new(x: u64, b: bool) -> Self {
        Self {
            bar: x,
            baz: b,
        }
    }

    fn is_baz_true(self) -> bool {
        self.baz
    }

    fn inc_bar(ref mut self) {
        self.bar += 1;
    }
}
```

Traits

```
trait SomeTrait {
    fn some_method(self) -> bool;
    fn some_associated_function() -> bool;
}

impl SomeTrait for SomeStruct {
    fn some_method(self) -> bool {
        . . .
    }

    fn some_associated_function() -> bool {
        . . .
    }
}
```

From Trait

```
trait From<T> {
    fn from(val: T) -> Self;
}

impl From<u8> for u64 {
    fn from(val: u8) -> u64 {
        val.as_u64()
    }
}
```

Constants and Associated Constants

```
const MODULE_CONSTANT: u64 = 42;

impl SomeStruct {
    const ASSOCIATED_CONSTANT: u64 = 84;

    fn some_method(self) {
        let _ = MODULE_CONSTANT;
        let _ = Self::ASSOCIATED_CONSTANT;
    }
}
```

Error Types, Result<T, E>, and panic

```
#[error_type]
enum SomeError {
    #[error(m = "First error has happened.")]
    FirstError: (),
    #[error(m = "Second error has happened.")]
    SecondError: (),
}
```

Error Types, Result<T, E>, and panic

```
fn get_value() -> Result<u64, SomeError> {
    if . . . {
        return Err(SomeError::FirstError);
    }

    . . .

    Ok(42)
}
```

Error Types, Result<T, E>, and panic

```
let val = match some_fn() {
    Ok(val) => val,
    Err(err) => panic err,
}
```