Nyckelbegrepp

Currying

https://en.wikipedia.org/wiki/Currying

https://www.youtube.com/watch?v=iZLP4qOwY8I&

https://wiki.haskell.org/Currying

When a function doesn't take all of it's arguments upfront. Instead, it wants you to give the first argument. Then the function returns another function, which you are supposed to call with the second argument. And so on, until all the arguments have been provided. The function at the end of the chain will be the one that returns the value that you actually want.

The idea is that your function can pass through the application and gradually recieve the argument that it needs.

```
f :: a -> (b -> c)
```

is the **curried** form of

```
g :: (a, b) -> c
```

Datatyp

https://en.wikipedia.org/wiki/Data_type

An attribute of data which tells the compiler or interpreter how the programmer intends to use the data. For example: int, bool etc.

Evalueringsstrategi

https://en.wikipedia.org/wiki/Evaluation_strategy

Evaluation strategies are used by programming languages to determine when to evaluate the argument(s) of a function call and what kind of value to pass to the function.

Purely functional languages like Haskell, as well as non-purely functional languages like R, use **call by need**. This means, if the function argument is evaluated, that value is stored for subsequent uses. If the argument is side-effect free, this produces the same results as call by name, saving the cost of recomputing the argument. Haskell is a well-known language that uses call-by-need evaluation. Because evaluation of expressions may happen arbitrarily far into a computation, Haskell only supports side-effects (such as mutation) via the use of monads. This eliminates any unexpected behavior from variables whose values change prior to their delayed evaluation.

Högre ordningens funktion

https://en.wikipedia.org/wiki/Higher-order_function

Takes one or more functions as arguments, and/or returns a function as its result.

In the following example, the higher-order function `twice` takes a function, and applies the function to some value twice. If `twice` has to be applied several times for the same `f` it preferably should return a function rather than a value.

```
twice :: (a -> a) -> (a -> a)
twice f = f . f

f :: Num a => a -> a
f = subtract 3

main :: IO ()
main = print (twice f 7) -- 1
```

Lambdakalkyl

https://en.wikipedia.org/wiki/Lambda_calculus

https://www.youtube.com/watch?v=9Q-Bi-Cg9tI

A formal system in mathematical logic for expressing computation based on function abstraction and application using variable binding and substitution. It is a universal model of computation that can be used to simulate any Turing machine.
”Describes functions with Lambda expressions.”

A lambda expression has a head and a body: λ(head).(body)

Head hols the function parameters, and the body describes what is returned by the fuction.

Example: λ(ab.aqbr) c d = cqdr AND λ(ab.aqbr) d c = dqcr

Boolean algebra is to computer science what Lambda calculus is to functional programming. It plays an important role in the design and base of Haskell..

Lat evaluering

https://en.wikipedia.org/wiki/Lazy_evaluation

Lazy evaluation AKA **call-by-need** is an evaluation strategy which Haskell uses, that delays the evaluation of an expression until its value is needed (non-strict evaluation) and which also avoids repeated evaluations. Can reduce running time.

Mönsterpassning

https://en.wikipedia.org/wiki/Pattern_matching

http://learnyouahaskell.com/syntax-in-functions

Pattern matching consists of specifying patterns to which some data should conform and then checking to see if it does and deconstructing the data according to those patterns.

```
lucky :: (Integral a) => a -> String
lucky 7 = "LUCKY NUMBER SEVEN!"
lucky x = "Sorry, you're out of luck, pal!"
```

When you call lucky, the patterns will be checked from top to bottom and when it conforms to a pattern, the corresponding function body will be used. The only way a number can conform to the first pattern here is if it is 7. If it's not, it falls through to the second pattern, which matches anything and binds it to x.

Polymorfism

https://en.wikipedia.org/wiki/Polymorphism_(computer_science)

A polymorphic function can be applied on several different types.

Ad hoc polymorphism

Different operators have different implementations depending on their arguments.

Example:

Imagine an operator `+` that may be used in the following ways:

1. `1 + 2 = 3`
2. `3.14 + 0.0015 = 3.1415`
3. `1 + 3.7 = 4.7`
4. `[1, 2, 3] + [4, 5, 6] = [1, 2, 3, 4, 5, 6]`
5. `[true, false] + [false, true] = [true, false, false, true]`
6. `"bab" + "oon" = "baboon"`

Parametric polymorphism

Using parametric polymorphism, a function or a data type can be written generically so that it can handle values identically without depending on their type.

For example, a function `append` that joins two lists can be constructed so that it does not care about the type of elements: it can append lists of integers, lists of real numbers, lists of strings, and so on. Let the *type variable a* denote the type of elements in the lists. Then `append` can be typed

```
forall a. [a] × [a] -> [a]
```

where `[a]` denotes the type of lists with elements of type *a*. We say that the type of `append` is *parameterized by a* for all values of *a*. (Note that since there is only one type variable, the function cannot be applied to just any pair of lists: the pair, as well as the result list, must consist of the same type of elements.) For each place where `append` is applied, a value is decided for *a*.

Subtyping

When a name denotes instances of many different classes related by some common superclass

Referenstransparens

https://en.wikipedia.org/wiki/Referential_transparency

An expression is called referentially transparent if it can be replaced with its corresponding value without changing the program's behavior. This requires that the expression is pure.

Ren funktion

https://en.wikipedia.org/wiki/Pure_function

the expression value must be the same for the same inputs and its evaluation must have no side effects.

Thus a pure function is a computational analogue of a mathematical function.

Överlagring

Method overloading

https://en.wikipedia.org/wiki/Method_overriding

the ability to create multiple functions of the same name with different implementations

Operator overloading

https://en.wikipedia.org/wiki/Operator_overloading

The same as ad hoc polymorphism (see above). You can define a user-defined type and overload a operator. (+ can be used with different types, which will mean different things).

Överskuggning

https://en.wikipedia.org/wiki/Method_overriding

Method overriding, in object-oriented programming, is a language feature that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its superclasses or parent classes.