Home
Course page
Module descriptor
Exercises
Last year's course
Lecture videos
Lecture slides
Tutorials
Labs
Study class 1

MISC
The time
Edi. weather
Gla. weather
Edi. sun
Gla. sun
Edi. wind
Gla. wind
For martians
Google link converter
Cycle route
Tatoeba
Translate
Linguee

Programming Languages (F28PL)

As a matter of good practice, code must be written in `monospace font` (also called `typewriter font`), and be nicely justified and indented.

All these questions count towards your final exam marks, unless they are explicitly labelled as *Unmarked*. You can send answers to labhelpers as you do them; you do not need (and are not expected to have done) all the questions, before having your answers marked.

Deadlines are as follows:

Questions A and B
: End of Thursday 6 October (passed).

Questions C and D
: End of Saturday 15 October (passed).

Questions E and F
: End of Thursday 27 October (passed).

Questions G, H, and I
: End of Saturday 5 November.

Question J
: End of Thursday 10 November.

Questions K and Z
: End of Thursday 24 November.

If you have not already done so, you should consult my guide to how to answer a programming question, in which the most profound secrets of how to answer a programming question are revealed.

Question A (Basic SML).

Write SML functions to:

1. Multiply real `x` by integer `y` to give a real.
```
- realIntMult 3.3 3;
> 9.9 : real
```
2. Check if string `s1` has more letters than string `s2`.
```
- moreLetters "big" "small";
> false : bool
```
3. Check if character `c` represents a digit. Do not answer this by writing just `Char.isDigit`. Also, **do not hard-code** ASCII numbers such as '37' or '42'. This is bad practice, since most readers do not know ASCII encoding off by heart so it is unclear to the reader what represent.
```
- isDigit #"7";
> true : bool
```
4. If character `c` represents a digit then return its integer equivalent; otherwise return `~1`. Again, your answer should not hard-code ASCII numbers.
```
- digitValue #"a";
> ~1 : int
- digitValue #"7";
> 7 : int
```
5. Convert a real `r` into a tuple of its value and integer equivalent.
```
- conv 99.99;
> (99.99,99) : real * int
```
6. Implement the NAND function from the truth-table below. *Your implementation should not use the ML primitives* `not`, `andalso`, *or* `orelse`. *We are looking for you to write a program that is almost identical to the specification of the problem.*

X	Y	X NAND Y
FALSE	FALSE	TRUE

FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	FALSE

```
- NAND false true;
> true : bool
```

Question B (Recursion, chaining functions).

1. Find the integer logarithm to the base 2 of an integer n by repeatedly dividing by 2 and counting:
   ```
   log2 1 = 0
   log2 n = 1+log2 (n div 2)
   - log2 8;
   > 3 : int
   ```
2. Find the square root of an integer x by repeatedly decrementing an initial guess s:
   ```
   sqrt x s = s if s*s <= x
   sqrt x s = sqrt x (s-1) if s*s > x
   - sqrt 10 5;
   > 3 : int
   ```
3. Find the sum of the squares of the first n integers
   $n^2 + (n-1)^2 + (n-2)^2 + \ldots 1^2$.
   1. The sum of the squares of the first 0 integers is 0.
   2. The sum of the squares of the first n integers is n squared plus the sum of the squares of the first n-1 integers.
      ```
      - sumSq 3;
      > 14 : int
      ```
4. Find the sum of the halves of the first n integers $n/2 + (n-1)/2 \ldots + 1/2$.
   ```
   - sumHalf 6;
   > 9 : int
   ```
5. Find the sum of applying function f to the first n integers.
   1. The sum of applying f to the first 0 integers is 0.
   2. The sum of applying f to the first n integers is f applied to n plus the sum of applying f to the first n-1 integers.
      ```
      - fun inc x = x+1;
      > val inc = fn : int -> int
      - sumF inc 3;
      > 9 : int i.e. inc 3+inc 2+ inc 1+ 0
      ```
6. Write `sumSq` using `sumF`.
7. Write `sumHalf` using `sumF`.

Question C (Lists).

1. Write a function to drop the first n elements of a list l:
   ```
   drop n l : int -> 'a list -> 'a list
   drop 3 [1,2,3,4,5] ==> [4,5]
   ```
   - To drop 0 elements from a list return the list.
   - To drop n elements from the empty list, return the empty list.
   - To drop n elements from a list, drop n-1 elements from the tail.
2. Write a function to take the first n elements of a list l:
   ```
   take n l : int -> 'a list -> 'a list
   take 3 ["a","b","c","d","e"] ==> ["a","b","c"]
   ```
   - To take 0 elements from a list return the empty list.
   - To take n elements from the empty list, return the empty list.
   - To take elements from a list, put the head of the list onto the result of taking n-1 elements from the tail.
3. Write a function to check if list l1 starts list l2:
   ```
   starts l1 l2 : ''a list -> ''a list -> bool
   starts [1,2,3] [1,2,3,4] ==> true
   ```

- An empty list starts a list.
- A list does not start an empty list.
- A list starts a second list if they have same head and the tail of the first starts the tail of the second.

4. Write a function to check if list l1 is contained in list l2:
```
contains l1 l2 : 'a list -> 'a list -> bool
contains ["d","e","f"] ["a","b","c","d","e","f","g","h"] ==> true
```
   - A list is not contained in an empty list.
   - A list is contained in a second list if it starts the second list.
   - Otherwise, a list is contained in a second list if it is contained in the tail of the second list.
5. Write a function to delete a list from another list:
```
delete l1 l2 : 'a list -> 'a list -> 'a list
delete [3,4,5] [1,2,3,4,5,6] ==> [1,2,6]
```
   - To delete a list from the empty list, return the empty list.
   - To delete a list from a second list, if the first list starts the second list then drop the length of the first list from the second list.
   - Otherwise, put the head of the second list onto the result of deleting the first list from the tail of the second.
6. Write a function to delete every occurrence of a list from another list:
```
deleteAll l1 l2 : 'a list ->'a list -> 'a list
deleteAll [1,2,3] [3,2,1,2,3,2,1,2,3] ==> [3,2,2]
```
   - All occurrences have been deleted from an empty list.
   - If the first list starts the second list, delete it and then delete all occurrences in the remaining list.
   - Otherwise, put the head of the second list on the front of deleting all occurrences of the first list in the tail.

Question D (Essay-style, functional programming).

1. Explain *recursion*.
2. Explain *tail recursion* and how it differs from plain old recursion. If you want to write efficient code, which is better, and why?
3. Explain the *Church-Rosser property*, and how it is relevant to programming language execution.
4. The cost of running a calculation is calculated as follows: $programmercosts + computingcosts = totalcosts$. A more detailed equation is $designcosts + programmercosts + debuggingcosts + computingcosts = totalcosts$. With these equations in mind, describe some of the trade-offs involved in choosing a programming language for an embedded chip in a car versus an Android app.
5. "Pure functional programming has no global state."
   1. Explain what this assertion means.
   2. Give one advantage and one disadvantage of using a language without global state.
6. Explain the different optimisations possible in imperative versus declarative programming, and the implications for space and time efficiency and development cost.

Question E (Higher-order functions).

1. Write a function of type `bool -> bool -> bool`.

2. How many different (in the sense of calculating different truth-tables) functions are there in `bool -> bool -> bool`?
3. Explain the similarities and differences between `fn (x:int) => x` and `fn (x:real) => x`.
4. Write two functions of type `('a -> bool) -> 'a list -> bool`.
   - The first should return `true` when the first argument is true of *all* elements in the second argument;
   - the second should return `true` when the first argument is true of *some* element in the second argument.
5. Write a function of type `'a -> unit`. Is it the only one?
6. Write a function of type `'a -> ('a*'a)`. Is it the only one?
7. Write a function of type `'a -> 'b -> ('a*'b)`.
8. How many different functions are there in `'a -> 'b -> ('a*'b)`?
9. Write a function of type `'a -> 'a -> ('a*'a)`.
10. How many different functions are there in `'a -> 'a -> ('a*'a)`?
11. Write a function of type `(('a -> 'b)*'a) -> 'b`.
12. How many different functions are there in `(('a -> 'b)*'a) -> 'b`?
13. Write a function of type `('a -> 'b) -> ('b -> 'c) -> ('a -> 'c)`.
14. How many different functions are there in `('a -> 'b) -> ('b -> 'c) -> ('a -> 'c)`?
15. Write a function of type `('a -> ('b -> 'c)) -> ('a -> 'b) -> ('a -> 'c)`.
16. (Hard) Using exceptions or otherwise, write a function of type `(unit -> 'a)`.

Example Q & A.

Write a function of type *`bool -> bool`***.**

Model answer (clarification in italics)

The obvious two are `fn (x:bool) => x` and `fn (x:bool) => true`.
`fn x => x` and `fn x => true` are incorrect: they have type `'a -> 'a` and `'a -> bool`.

Is it the only one? If so, why? If not, how many are there?

Model answer

Many different programs of type `bool -> bool` exist, such as (I drop type annotations) `fn x => true` and `fn x => if true then true` and `fn x => (x=x)` and `fn x => true`; however, these all implement the same function "map `x` to `true`".
There are four: `fn (x:bool) => true`, `fn (x:bool) => false`, `fn (x:bool) => x`, and `fn x => not x`.

Describe all functions of type *`('a -> 'a) -> ('a -> 'a)`***.**

Model answer

So we're given `f:'a -> 'a` and we're given `x:'a` and we want to make something of type `'a`. Well this is not so hard: all we can do is apply `f` to `x` some finite (and possibly zero) number of times. So we have `fn f => fn x => x`, `fn f => fn x => (f x)`, `fn f => fn x => (f (f x))`, and so on. *Note how we follow the types!*

Question F (Strings and chars).

1. Write a function to remove all the non-letters off the front of a list of characters:
   `strip [#".",#" ",#"t#,#"h",#"e"] ==> [#"t",#"h",#"e"]`.
   - If the list is empty, return the empty list.
   - If the list starts with a letter, return the list.
   - Otherwise remove all the non-letters from the start of the tail of the list.
2. Write a function to take the next word off a list of characters. The function returns a tuple of the word as a list of characters and the rest of the list.
   `next [#"t",#"h",#"e",#" ",#"c",#"a",#"t",#" "…] ==>`

`([#"t",#"h",#"e"],[#" ",#"c",#"a",#"t",#" " ...]) : char list * char list`.

- If the list is empty, return a tuple of two empty lists.
- If the list starts with a non-letter, return a tuple of the empty list and the list.
- Otherwise, take the rest of the next word from the tail of the list returning a tuple of the rest of the next word and the rest of the tail of the list, put the head of the original list, which is the head of the word, onto the front of the rest of the word, and return the whole word list and the rest of the tail of the list.
  Hint: use a let with a tuple pattern to match the tuple for the rest of word and rest of tail of list.

3. Write a function to turn a list of characters into a list of individual words as strings.
   `words [#"t",#"h",#"e",#" ",…] ==>`
   `["the","cat","sat","on","the","mat"] : string list`
   - If the list is empty, return an empty list.
   - Otherwise, get the first word from the stripped list as a list of characters and the rest of the list, get the list of words from the rest of list and put the imploded first word on the front of the word list.
     Hint: use a local definition with a tuple pattern to match the first word and rest of list.
4. Write a function to increment the count for a word in a list of words and counts.
   `incCount "sat" [("cat",1),("the",1)] ==> [("cat",1),("sat",1), ("the",1)] : (string * int) list`
   `incCount "the" [("cat",1),("mat",1),("on",1),("sat",1),("the",1)] ==> [("cat",1),("mat",1),("on",1),("sat",1),("the",2)] : (string * int) list`
   - To increment the count for a word in an empty list, create a list with a tuple for the word and a count of 1.
   - To increment the count for a word in a list, if the word is the word in the first tuple, return the list with the first tuple's count incremented.
   - Otherwise if the word comes before the word in the first tuple in the list, add a new tuple for the word and a count of 1 on the front of the list.
   - Otherwise, put the head of the list on the front of the result of incrementing the count for the word in the tail of the list.
5. Write a function which given a list of words and an empty list, constructs the frequency count list.
   `counts ["the","cat","sat","on","the","mat"] [] ==> [("cat",1), (mat",1),("on",1),("sat",1),("the",2)];`
6. Write a function which given a file name, opens the file, inputs the whole file as a string, explodes the string, makes the word count list, closes the file and returns the word count list.
7. Write a function which given a word count list displays it on standard output as a table of words and counts.

Example Q & A.

Write a function `twoFalse` which given a list of booleans, returns `true` if the list has two or more elements and the first two elements are `false`, and `false` otherwise.

Model answer

Convert the question directly into ML code:

`fun twoFalse (false::false::_) = true | twoFalse _ = false.`

Cut-and-paste into the compiler to make sure and run it on a couple of test cases, but it's obviously correct because it's just the statement of the question.

Here are some bad answers, in increasing order of cosmic yeuchness:

```
fun twoFalse x = if (hd x)=false and (hd(tl x))=false then true else false
```

```
fun twoFalse (x::y::tl) = if (not x) andalso (not y) then true |
```

`twoFalse x = false`
`fun twoFalse x = if (length x)=0 then raise Subscript else ...` (I don't like this: first, because the question doesn't say "raise an exception" so you're introducing behaviour that was not specified; and second, it's actually rather dangerous behaviour because your program passes flow of control to the innermost `Subscript` handler, which we know nothing about: e.g. it could have been hacked by the North Korean government.)

Question G (Programming).

1. Consider the representation of a binary number as a list of booleans, where the first boolean is the *least* significant digit. Thus, `[false,true]` represents two and `[false,true,false,true]` represents ten. Recall the school algorithm for addition, involving carrying ones over to more significant digits as appropriate.
   Implement this binary addition as a function of type `bool list -> bool list -> bool list`.
2. Using an accumulator variable or otherwise, write a function `max` of type `int list -> int` that returns the greatest element of its input.
3. Using accumulator variables or otherwise, write a function `avg` of type `real list -> real` that returns the average of a list.
4. (Harder) Using accumulator variables or otherwise, write a function `modes` of type `int list -> (int list*int)` that returns the list of those integers that occur most often in the input, along with the number of times they occur (see mean, median, and mode). For example `modes [5,2,5,3,3]` should return `([5,3],2)` or `([3,5],2)`.

Question H (Python lists, polymorphism, recursion, and reduce).

Make sure to test all your programs below, including on `[]`.

1. Using iteration (a for-next or a while loop or similar), write a program `mult1` that if given a list of integers or reals will return their product (the result of multiplying all the numbers together).
   We do not care about behaviour if `mult1` is not given a list of integers or reals; likewise for `mult2` and `mult3` below.
   `mult1` should return `1` for the empty list, because `1` is the **multiplicative unit**. Thus
   - `mult1([])` calculates `1`, and
   - `mult1([1,2,3,10])` calculates `60`.
2. Using recursion, write a program `mult2` that if given a list of integers or reals will return their product (warning: this question is very slightly harder than it seems; make sure to test your answer).
3. Using `range` or otherwise, write a one-line piece of code that breaks `mult2` (i.e. make it crash when it "should" return a value). Why did this happen?
4. Using Python lambda and `reduce` (see specifically the example on line 2 of this link), write a program `mult3` that if given a list of integers or reals, returns their product.
5. What do `mult1`, `mult2`, and `mult3` return if applied to `[1.0,2,3.0,10]`? What would corresponding programs do if they were written in ML and applied to such an input? What does this tell us about ML vs Python typing?
6. Order `mult1`, `mult2`, and `mult3` from slowest to fastest and explain your order. There is no single "right" answer here, and this is a theoretical question, not an empirical one; you are welcome to measure performance but what interests me is your understanding.

7. An issue with the programs above is that they only work for numbers (integers or reals), however, there is no reason in principle they should not also work with strings or lists (by concatenation). Note that in Python you can check whether a variable `x` has type `str` (for example) with `if type(x) is str: print("x is a string!")`, and similarly for lists `if type(x) is list: print("x is a list!")`.
   Write a program `multpoly` that outputs the product of a list of numbers, and the concatenation (i.e. chaining) of a list of strings or lists. So
   - `multpoly([1,2,3,10])` calculates `60`, and
   - `multpoly(["1","2","3","10"])` calculates `"12310"`, and
   - `multpoly([[1],[2,3],[10]])` calculates `[1,2,3,10]`.
     We do not care what `multpoly` calculates on the empty list. We do care about a clear and elegant implementation.
8. Using `if type(x) is list` or otherwise, write a **recursive** program `flatten` that inputs an arbitrarily nested list structure and outputs a "flattened" list consisting of the list of all the data in the list, but with any nested list structure removed. So for instance,
   - `flatten([])` returns `[]`, and
   - `flatten([["hi"],5])` returns `["hi",5]`, and
   - `flatten([[[],[["hello"],[" "]],[1],[[["world"]],[]]],"!"])` returns `["hello"," ",1,"world!"]`.

Question I (More Python)

1. Write a program `factors` that inputs a number `n`, which you may assume is greater than 2, and outputs the list of its prime factors (prime numbers are 2, 3, 5, 7, 11, 13, 17, and so on). You may assume basic arithmetic operations, including modulus `n % k` and divisibility. However, if you find a library call "prime_factors_of", you can't use that, sorry.
2. Write a program `largest` that inputs a nonempty list of numbers and outputs its greatest element. You do not need to worry about error-handling, e.g. if the input is the empty list.
3. Write a program `largest_factor` that inputs a number `n` and outputs its largest prime factor.
4. Write a program `firstbigfib` that inputs a number `n` and outputs the index of the first Fibonacci number to contain `n` digits.
5. Write a program `firstbigf` that inputs a number `n` and a function `f` mapping numbers to numbers, and outputs the least strictly positive number `i` (so `i` is in `[1,2,3,...]`) such that `f(i)` contains `n` digits. So for example if `f=lambda x:10**x` then `firstbigf f 10` should calculate `9` (because `10**9`, which is equal to `1000000000`, has ten digits).
6. Suggest, in very general terms, how we might optimise `firstbigf` to run on a multicore architecture, with a roughly m-fold speedup where m is the number of cores.
7. A **Pythagorean triple** is a 3-tuple of strictly positive numbers `(x,y,z)` such that `z*z=x*x+y*y`. Using Python generators (page 97 of these slides or the generator that yields items example here), write a Python generator function `triples()` to generate all Pythagorean triples. So:
   - `x=triples()` and then `next(x)` should return a first triple, and then
   - `next(x)` should return a second triple, and then
   - `next(x)` should return a third one, and so on.
     Test your program further by running it on `list(itertools.islice(triples(), 1, 300))` (using the Internet or otherwise, make sure you understand what this does; you may need to `import itertools`).

Question J (Python lists, sets, and dictionaries)

1. State what the following Python programs calculate, and explain why:
   1. `{ 0:"zero", 1:"one", 2:"two" }[1]`
   2. `range(1000)[0]`
   3. `for x in range(1,1000): print(list(enumerate(range(x)))[0][0])`
   4. `"hello world"[:-2]`
2. Explain the difference in behaviour between the following two code snippets:
```
x=[""] # Snippet 1
for i in x: x.append("")
x=[""] # Snippet 2
for i in x[:]: x.append("")
```
3. Explain the behaviour of the following code snippet:
```
x=[[]] ; x[0].extend(x)
while True:
    print(x)
    x = x[0]
```
4. The following Python program returns an error.
   `{[0]:0}[[0]]`
   1. Explain what the error is and why it arises.
   2. Suggest how to fix it.
5. Using list comprehensions and lambda, write a program that if given a number `n`, calculates the first `n` even numbers (starting from `0`).
6. Using list comprehensions and lambda, write a program that if given a list of numbers `l`, will return the sublist of those numbers in `l` that are divisible by 3.
7. The `zip` function inputs two lists (which you may assume have the same length) and outputs the list of pairs of the lists 'zipped' together. For instance, `zip([1,2,3], ["one","two","three"])` returns `[(1,"one"),(2,"two"),(3,"three")]`. Implement `zip` yourself, from first principles, using list comprehension (so **not** using a for-next loop and the append method; too easy).
8. The following Python programs calculates the first ten squares and the first ten pentagonal numbers:
```
squares = [x*x for x in range(10)]
pentagonals = [x*(3*x-1)//2 for x in range(10)]
```
   (For this question, `0` is a square and a pentagonal number.)
   Based on those examples, and using list comprehensions if possible, write a Python program that calculates a list of numbers that are both square and pentagonal (up to some computational bound, i.e. the program is allowed to stop after a while). Test your program by calculating the first four such numbers.
9. Nonnegative integers can be implemented in Python by `nest(0)=[]` and `nest(n+1)=[nest(n)]`. Write programs `nest` and `unnest` which translate between nonnegative integers and the nesting implemenation.
   We do not care what happens at incorrect input. Implement addition and multiplication as direct functions on this implementation (i.e. not by untranslating, adding the numbers, and retranslating).
   (The student who finds this kind of thing amusing, might be interested in the surreal numbers.)

Question K (Prolog)

1. Write Prolog clauses for the following scenario:
   1. Jake is a person
   2. Jill is a person
   3. John is a person

4. Joan is a person
5. Jake likes tomato
6. Jill likes cheese
7. Jill likes tomato
8. John likes cheese
9. Joan likes cheese
10. Joan likes tomato
11. Jake knows Jill
12. Jill knows John
13. John knows Joan
14. Joan knows Jake
15. Every person knows themself
16. If a person likes cheese and that person likes tomato then they like pizza

2. Test your program by writing questions to check:
   1. Whether Jake likes pizza
   2. Which persons like pizza
   3. Which persons know other persons who likes pizza
3. Write Prolog clauses for the following scenario:
   1. Aberdeen is a place
   2. Dundee is a place
   3. Edinburgh is a place
   4. Glasgow is a place
   5. Kirkcaldy is a place
   6. St Andrews is a place
   7. There are 60 miles between Aberdeen and Dundee
   8. There are 60 miles between Dundee and Edinburgh
   9. There are 45 miles between St Andrews and Edinburgh
   10. There are 10 miles between Dundee and St Andrews
   11. There are 60 miles between Dundee and Aberdeen
   12. There are 30 miles between St Andrews and Kirkcaldy
   13. There are 35 miles between Kirkcaldy and Edinburgh
   14. There are 45 miles between Glasgow and Edinburgh
   15. The miles from a first place to a second place is the miles between the first place and the second place or the miles between the second place and the first place
   16. The distance between a first and second place is the miles from the first to the second or the miles from the first to a third place plus the distance between the third and second place
4. Test your program by writing questions to find distances between:
   1. Edinburgh and St Andrews
   2. Aberdeen and Glasgow.

Note that you are not required to identify different routes between these pairs of places.

Question Z

1. (Reasonable) The Bubblesort and Quicksort algorithms all have type `int list -> int list`.
   1. Implement Bubblesort in ML.
   2. Implement Quicksort in ML.
2. (Hard) Write a pair of functions `tofun : int -> ('a->'a)` and `fromfun : ('a->'a) -> int` such that `(fromfun o tofun) n` evaluates to `n` for every `n:int`. Do not use exceptions.
3. (Infernal) Write a function of type `(('a -> 'b) -> 'b) -> 'a`.
4. (Very hard) Implement the Tower of Hanoi as a function of type `unit -> (int list*int list*int list)`.

5. (Very hard) What does this program calculate, and how?

```
f = lambda x: [[y for j, y in enumerate(set(x)) if (i >> j) & 1] for
i in range(2**len(set(x)))]
```