1. I define a type `Literal` in my code to represent integers, booleans and None.

```
export type Literal<A> =
    { a ?: A, tag: "num", value: number }
  | { a ?: A, tag: "bool", value: boolean }
  | { a ?: A, tag: "none" }
```

In WASM, integers are their real value, booleans are 0 or 1 and None is 0.

To print True, False, I modified the function `print` and design three new function `print_num`, `print_bool` and `print_none`. In typeChecker, when checking the function `print` in python code, I transfered it to `print_num`, `print_bool` and `print_none` according to the type of the parameter.

```
print_num: (arg : any) => {
          const elt = document.createElement("pre");
          document.getElementById("output").appendChild(elt);
          elt.innerText = arg;
          return arg;
        },
        print_bool: (arg : any) => {
          const elt = document.createElement("pre");
          document.getElementById("output").appendChild(elt);
          elt.innerText = arg === 1 ? "True" : "False";
          return arg === 1 ? "True" : "False";
        },
        print_none: (arg : any) => {
          const elt = document.createElement("pre");
          document.getElementById("output").appendChild(elt);
          elt.innerText = "None";
          return "None";
        },
```

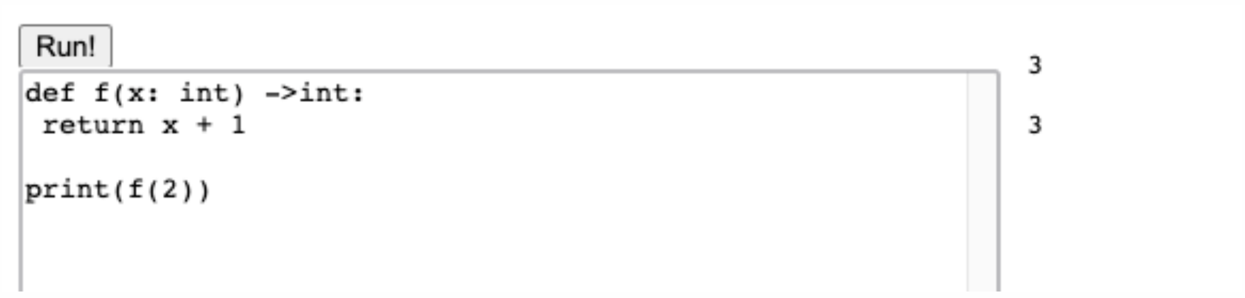
```
case "builtin1":
            const arg1 = expr.arg;
            const typedArg1 = typeCheckExpr(arg1, env);
            if (expr.name === "print"){
                if (typedArg1.a === Type.int) {
                    return { ...expr, name: "print_num", a: Type.none, arg: typedArg1};
                } else if (typedArg1.a === Type.bool) {
                    return { ...expr, name: "print_bool", a: Type.none, arg: typedArg1};
                } else {
                    return { ...expr, name: "print_none", a: Type.none, arg: typedArg1};
                }
            }else if (expr.name === "abs"){
                return { ...expr, a: Type.int, arg: typedArg1};
            }else{
                throw new Error("REFERENCE ERROR");
            }
```

2.

- At least one global variable

  I'm sorry that the global variable is not currently supported.

- At least one function with a parameter



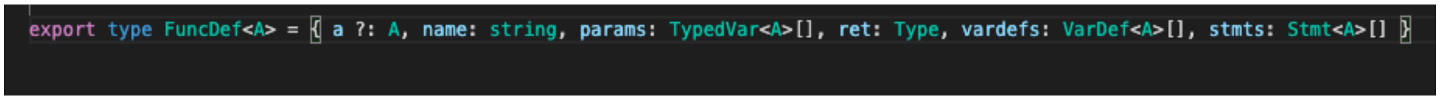
```
def f(x: int) ->int:
 return x + 1

print(f(2))
```

```
3

3
```

The parameters are stored in the `FuncDef.params`. And `FuncDef` is defined here(https://github.com/PengWei98/pa1/blob/main/ast.ts#L40)

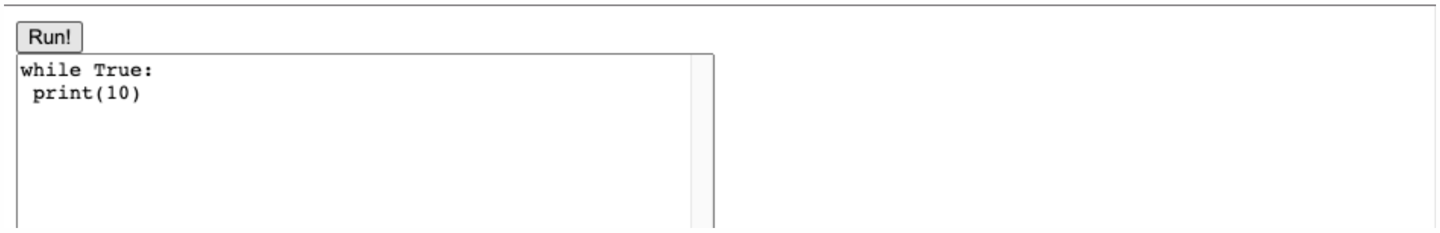


```
export type FuncDef<A> = { a ?: A, name: string, params: TypedVar<A>[], ret: Type, vardefs: VarDef<A>[], stmts: Stmt<A>[] }
```

- At least one variable defined inside a function

  I'm sorry that the variable defined inside a function is not currently supported.

3.

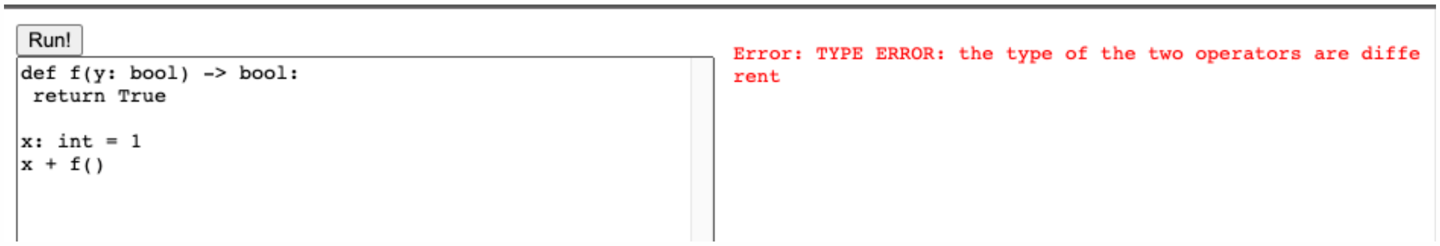


```
while True:
 print(10)
```

The integer was printed in the consoler continuously and finally the web browser broken down.

4.

- 1.

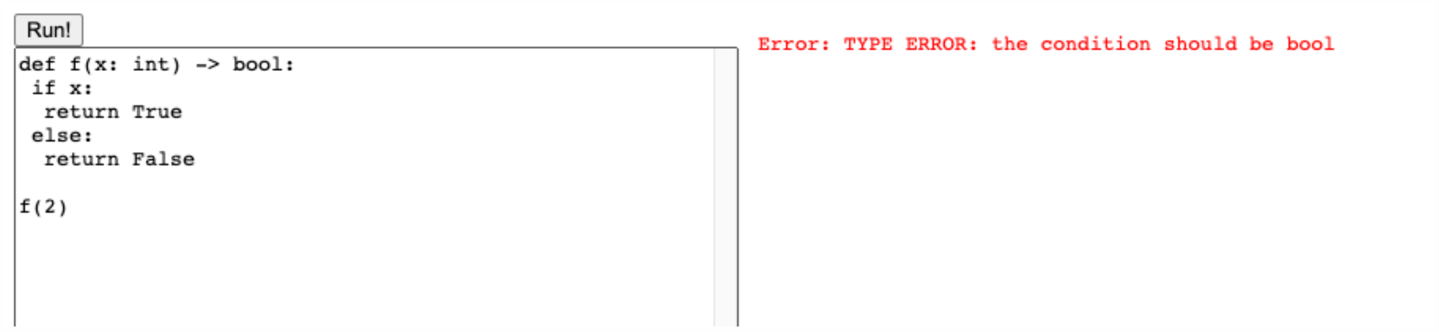


```
def f(y: bool) -> bool:
 return True

x: int = 1
x + f()
```

```
Error: TYPE ERROR: the type of the two operators are diffe
rent
```

- 2.

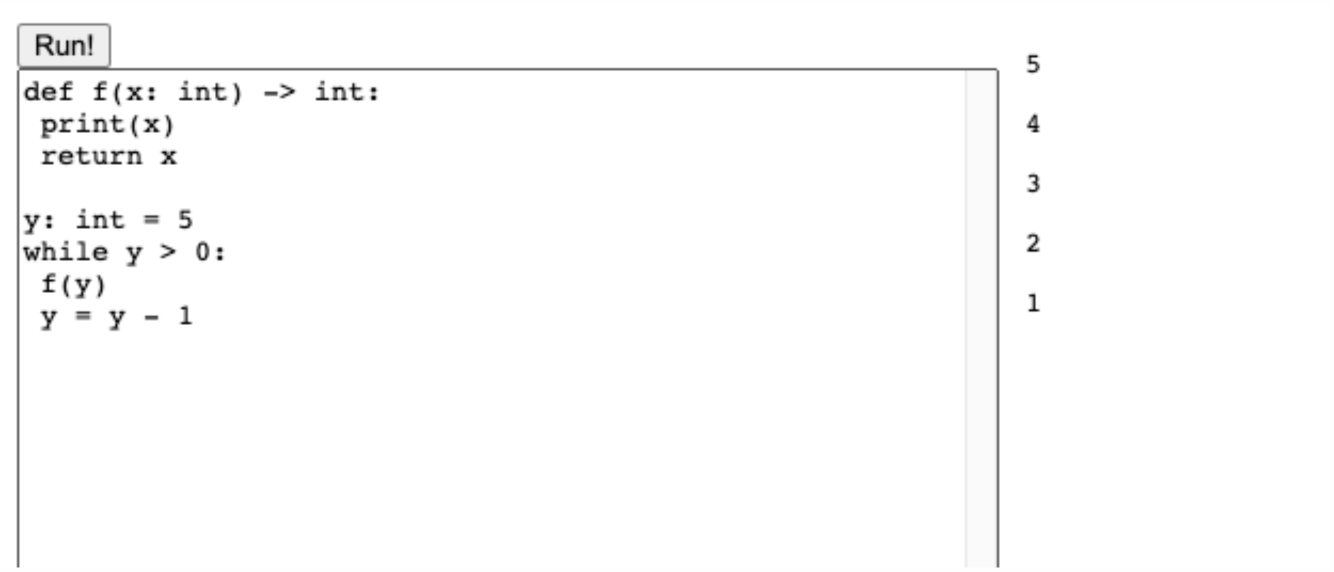


```
def f(x: int) -> bool:
 if x:
  return True
 else:
  return False

f(2)
```

```
Error: TYPE ERROR: the condition should be bool
```

- 3.

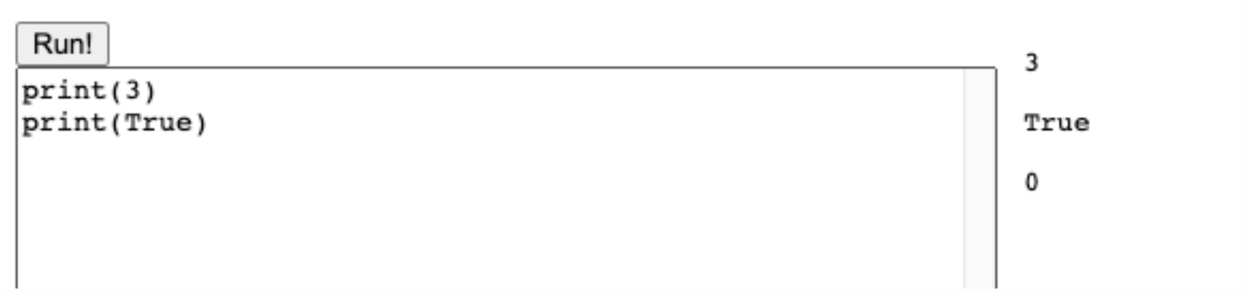


```
def f(x: int) -> int:
 print(x)
 return x

y: int = 5
while y > 0:
 f(y)
 y = y - 1
```

```
5
4
3
2
1
```

- 4. This function is not supported.
- 5.



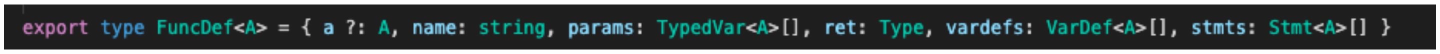
```
print(3)
print(True)
```

```
3
True
0
```

- 6.Recursive function is not supported.
- 7.Recursive function is not supported.

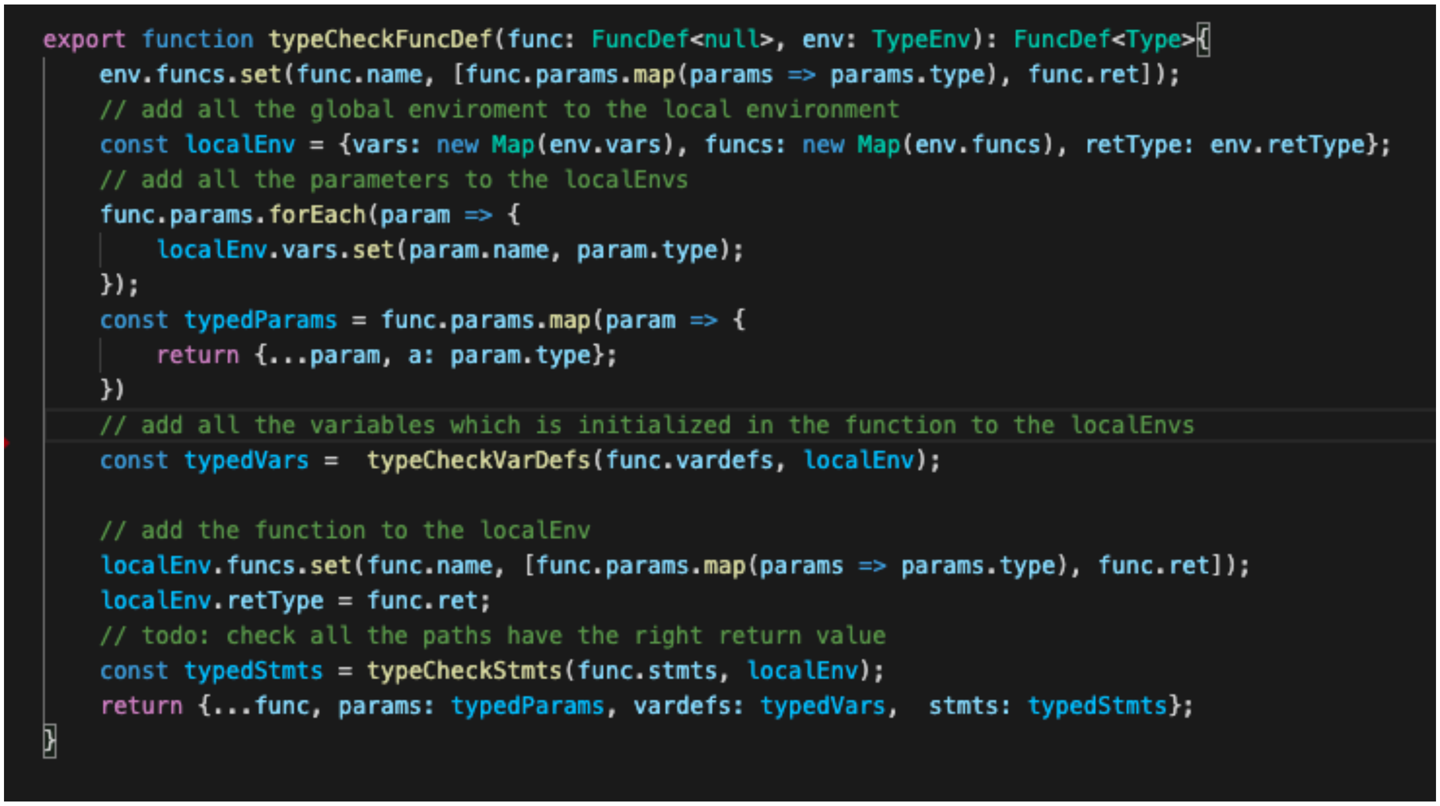
5. I choose the example in 2.2(Give an example of a program that uses at least one function with a parameter)

I define the type of the function like this, the `FuncDef.params` is its params, represented by a list. So it can have zero or more parameters.



```
export type FuncDef<A> = { a ?: A, name: string, params: TypedVar<A>[], ret: Type, vardefs: VarDef<A>[], stmts: Stmt<A>[] }
```

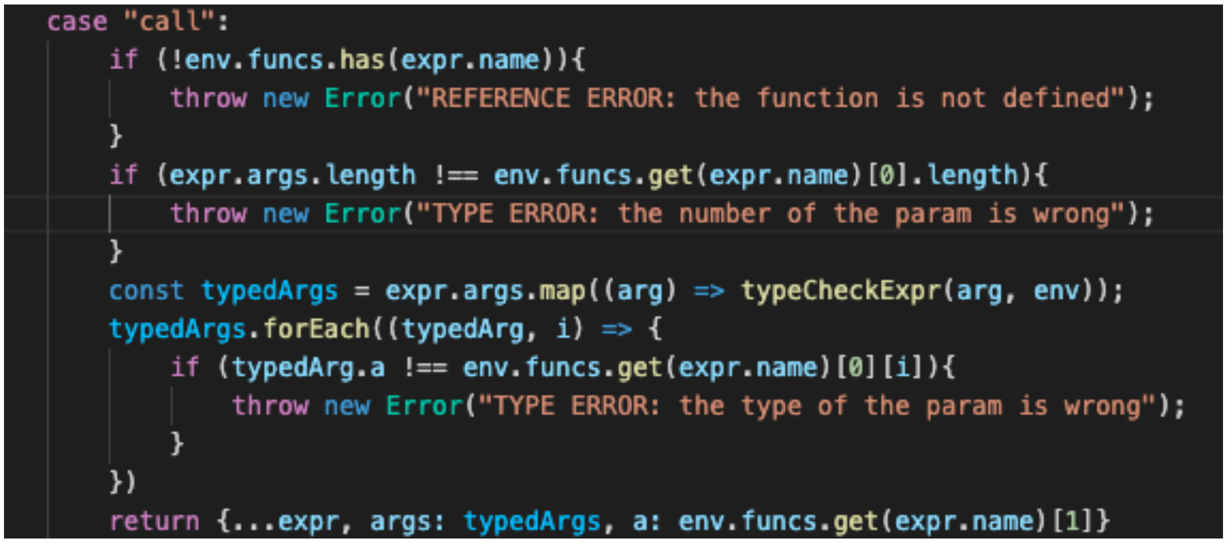
In the typechecker, I check the type of the function:



```
export function typeCheckFuncDef(func: FuncDef<null>, env: TypeEnv): FuncDef<Type>{
    env.funcs.set(func.name, [func.params.map(params => params.type), func.ret]);
    // add all the global enviroment to the local environment
    const localEnv = {vars: new Map(env.vars), funcs: new Map(env.funcs), retType: env.retType};
    // add all the parameters to the localEnvs
    func.params.forEach(param => {
        localEnv.vars.set(param.name, param.type);
    });
    const typedParams = func.params.map(param => {
        return {...param, a: param.type};
    })
    // add all the variables which is initialized in the function to the localEnvs
    const typedVars =  typeCheckVarDefs(func.vardefs, localEnv);

    // add the function to the localEnv
    localEnv.funcs.set(func.name, [func.params.map(params => params.type), func.ret]);
    localEnv.retType = func.ret;
    // todo: check all the paths have the right return value
    const typedStmts = typeCheckStmts(func.stmts, localEnv);
    return {...func, params: typedParams, vardefs: typedVars,  stmts: typedStmts};
}
```

When the function is called, I will choose the type for each arguments. And I also checked the whether the function has already defined.



```
case "call":
    if (!env.funcs.has(expr.name)){
        throw new Error("REFERENCE ERROR: the function is not defined");
    }
    if (expr.args.length !== env.funcs.get(expr.name)[0].length){
        throw new Error("TYPE ERROR: the number of the param is wrong");
    }
    const typedArgs = expr.args.map((arg) => typeCheckExpr(arg, env));
    typedArgs.forEach((typedArg, i) => {
        if (typedArg.a !== env.funcs.get(expr.name)[0][i]){
            throw new Error("TYPE ERROR: the type of the param is wrong");
        }
    })
    return {...expr, args: typedArgs, a: env.funcs.get(expr.name)[1]}
```

Explanation: Some of the functions are too hard for me. Acctually, for some features like resursive function or global variable, the code can be compliled into WASM but there are some runtime error when running WASM. And the resource on the internet about WASM are so rare so it's really hard for me to find the solutions. I will try my best to solve it.