Język algorytmiczny – podsum.

- Zmienna = nazwa + typ(?) + dana/wartość
  a=123; tab[i]=123; rek.pole1=123
  a=a+1; // to nie jest równanie mat
- Typy danych:
  - Proste: int, float, double, string(?), char, bool
  - Złożone: tablica, strukt/rekord, wskaźnik, referencja, obiekt
    /j. skryptowe/ lista, słownik, …
- Instrukcje sterujące:
  - Instr przypisania: zmienna=wartość; zm:=wart
  - If *warunek then kod1 else kod2*
    If (*warunek) then { kod1 } else { kod2 }*
  - While *warunek* do *kod*
    While (*warunek)* { *kod* }
  - For *zm=wart1* to *wart2* do *kod*
  - Repeat *kod* until *warunek*

Pętle...

W pseudo-kodzie wcięcia oznaczają bloki kodu

Język algorytmiczny - podsum

- Procedury i funkcje:
  proc/fun Proc1 (parametry) kod;
  funkcje: wynik zwracany przez „return wyr” w kodzie

```
function dodaj(a,b)
   return a+b;
function dodaj(a,b)
   dodaj= a+b; // alt sposób zwracania wartości
function max(a,b)
   if a>b then return a; else return b; // return kończy proc
```

- Zmienne globalne i lokalne w procedurach
  - Parametry i zm lokalne w kodzie proc
    każde wywołanie - inna ramka stosu (inna kopia zm!)
  - Zm globalne deklarowane poza/ przed proc

Język algorytmiczny - podsum

- Wyrażenia, operatory (w war, instr przyp, wywołania proc)
  d = a+b+c <=> d = (a+b)+c
- Priorytety, łączność operatorów

Język C

Tablica 2-1. Priorytety i łączność operatorów

Operatory	Łączność
() [] -> .	lewostronna
! ~ ++ -- + - * & (*typ*) sizeof	prawostronna
* / %	lewostronna
+ -	lewostronna
<< >>	lewostronna
< <= > >=	lewostronna
== !=	lewostronna
&	lewostronna
^	lewostronna
|	lewostronna
&&	lewostronna
||	lewostronna
?:	prawostronna
= += -= *= /= %= ^= |= <<= >>=	prawostronna
,	lewostronna

Jednoargumentowe operatory +, -, * oraz & mają priorytet wyższy niż ich odpowiedniki dwuargumentowe.

Inne przykłady:

a+b*c <=> a+(b*c)

a+1<b+1 <=>(a+1)<(b+1)

Kateg operatorów:

- arytmetyczne
- logiczne, porówania
- dostęp do danych
- przypisanie

A jak to wygląda w Python3 ?

Zmienne:

```
x=123; # int
x=[1,2,3]; # lista
print(x[0]); # dostęp do elem listy
x={'q':1, 'w':2}; # słownik
print(x['q']); # dostęp do elem słown
x=[1,2,[1,2,”q”]]; # lista zagnieżdż.
x=[1,2,{'q':1, 'w':2}];
  # ^ lista/ słownik zagnieżdzone
print( x[2]['w'] )
```

Procedury/ funkcje:

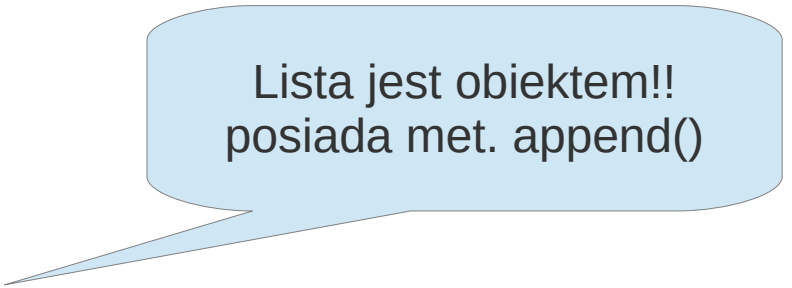
```
def dodaj(x,y):
  return(x+y)
def funkcja(x):
  if x<10:
    return('duża')
  else:
    return('mała')
```

Pętle:

```
x=[]
for i in range(1,5):
  x.append(i)
print(x)
# ^ wypisuje listę [1,2,3,4]
```

Pętle:

```
x=[]; i=1
while i<5:
  x.append(i); i=i+1;
print(x)
# ^ wypisuje listę [1,2,3,4]
```



A jak to wygląda w Python3 ?

Podział kodu na 'moduły”:

- Tworzymy plik **p1.py**
- Definiujemy w tym pliku procs
- Z konsoli importujemy moduł:
  import **p1**
- Proc z modułu wywołujemy tak:
  p1.fun1(1,2,3)
- dir(p1) pokazuje zawartość modułu
- Jak przeładować moduł ?
  ?? po modyfikacji pliku ??
  import importlib
  importlib.reload(**p1**)

Proc ze zm globalnymi

```
def powieksz(x):
  global zm
    # zm globalne muszą być
    # zadeklarowane w proc !!
  zm=zm+x

zm= 1000
powieksz(3)
print(zm)
  # wypisuje 1003
```