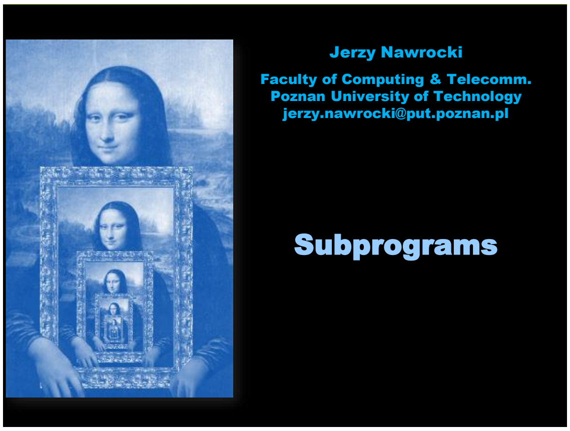
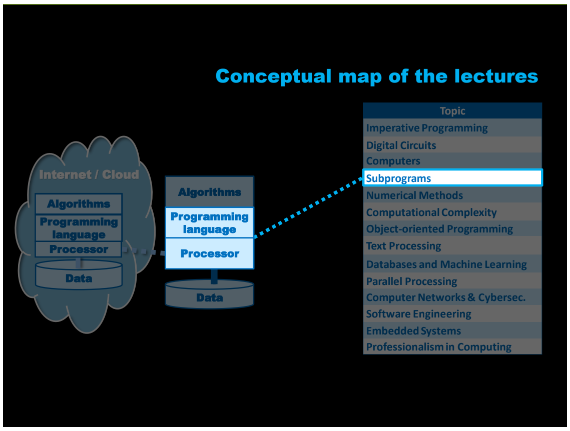


Jerzy Nawrocki

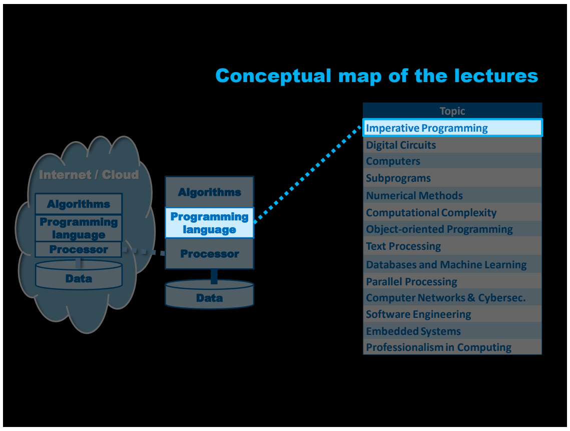
Faculty of Computing & Telecomm.
Poznan University of Technology
jerzy.nawrocki@put.poznan.pl

Subprograms



Conceptual map of the lectures

Topic
Imperative Programming
Digital Circuits
Computers
Subprograms
Numerical Methods
Computational Complexity
Object-oriented Programming
Text Processing
Databases and Machine Learning
Parallel Processing
Computer Networks & Cybersec.
Software Engineering
Embedded Systems
Professionalism in Computing



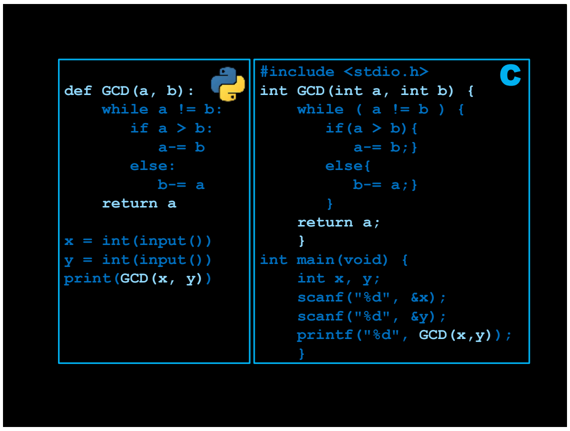
Conceptual map of the lectures

Topic
Imperative Programming
Digital Circuits
Computers
Subprograms
Numerical Methods
Computational Complexity
Object-oriented Programming
Text Processing
Databases and Machine Learning
Parallel Processing
Computer Networks & Cybersec.
Software Engineering
Embedded Systems
Professionalism in Computing

Agenda

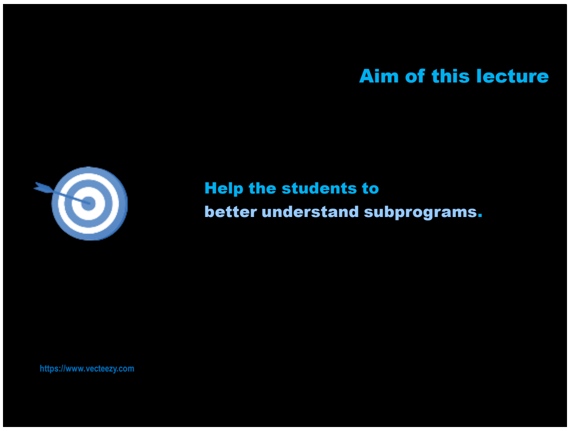
- Very simple programs
- Conditionals
- Iteration
- Simple functions
- Simple arrays

- Declarative programming



```
def GCD(a, b):
    while a != b:
       if a > b:
          a-= b
       else:
          b-= a
    return a

x = int(input())
y = int(input())
print(GCD(x, y))
```

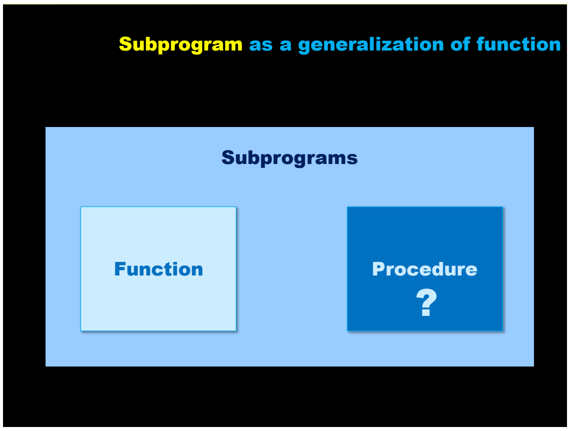
```
#include <stdio.h>
int GCD(int a, int b) {
    while ( a != b ) {
       if(a > b){
          a-= b;}
       else{
          b-= a;}
       }
    return a;
    }
int main(void) {
    int x, y;
    scanf("%d", &x);
    scanf("%d", &y);
    printf("%d", GCD(x,y));
    }
```



Aim of this lecture

Help the students to
better understand subprograms.

https://www.vecteezy.com

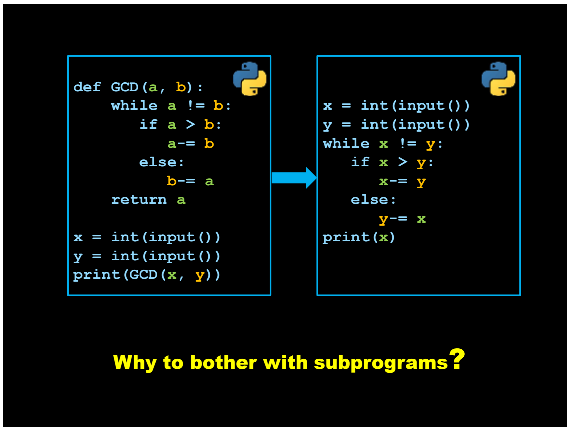


Subprogram as a generalization of function

Subprograms

Function

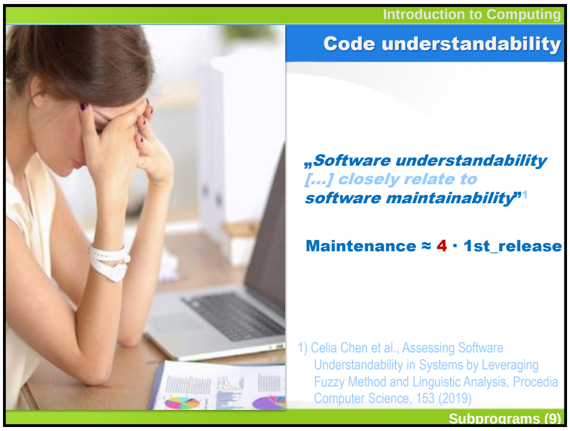
Procedure ?



```
def GCD(a, b):
    while a != b:
        if a > b:
            a-= b
        else:
            b-= a
    return a

x = int(input())
y = int(input())
print(GCD(x, y))
```

```
x = int(input())
y = int(input())
while x != y:
    if x > y:
        x-= y
    else:
        y-= x
print(x)
```

Why to bother with subprograms?



Introduction to Computing

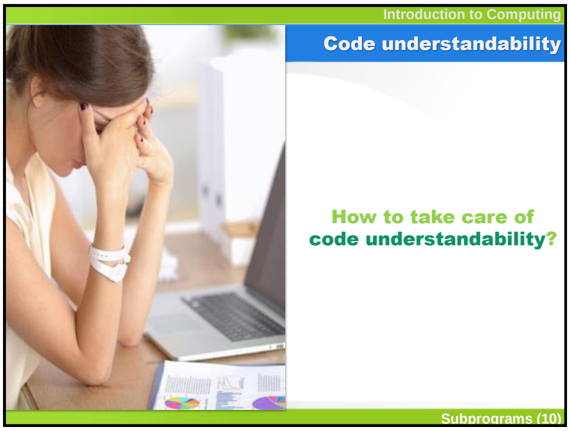
Code understandability

„Software understandability [...] closely relate to software maintainability"[1]

Maintenance ≈ 4 · 1st_release

[1] Celia Chen et al., Assessing Software Understandability in Systems by Leveraging Fuzzy Method and Linguistic Analysis, Procedia Computer Science, 153 (2019)

Subprograms (9)

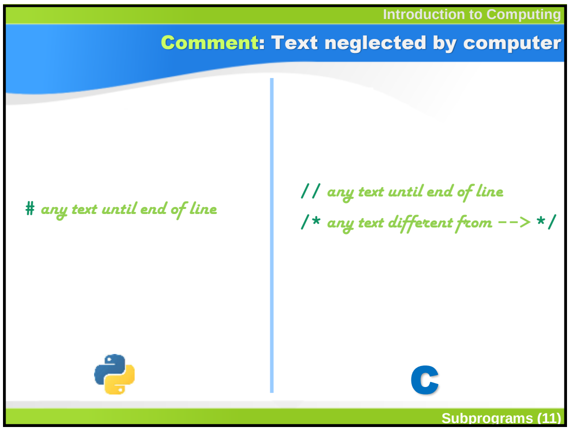


Introduction to Computing

Code understandability

How to take care of code understandability?

Subprograms (10)



Introduction to Computing

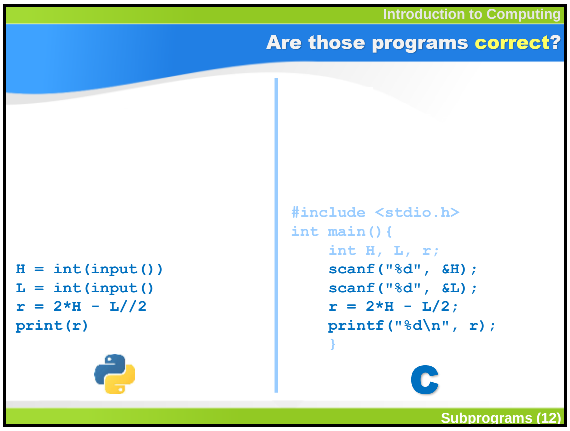
Comment: Text neglected by computer

```
# any text until end of line
```

```
// any text until end of line
/* any text different from --> */
```

C

Subprograms (11)



Introduction to Computing

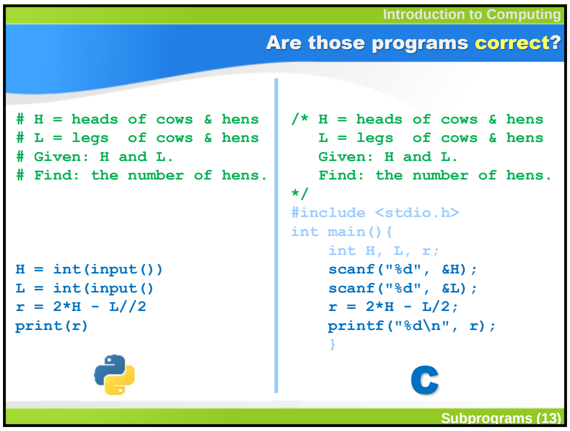
Are those programs correct?

```
H = int(input())
L = int(input()
r = 2*H - L//2
print(r)
```

```
#include <stdio.h>
int main(){
    int H, L, r;
    scanf("%d", &H);
    scanf("%d", &L);
    r = 2*H - L/2;
    printf("%d\n", r);
    }
```

C

Subprograms (12)

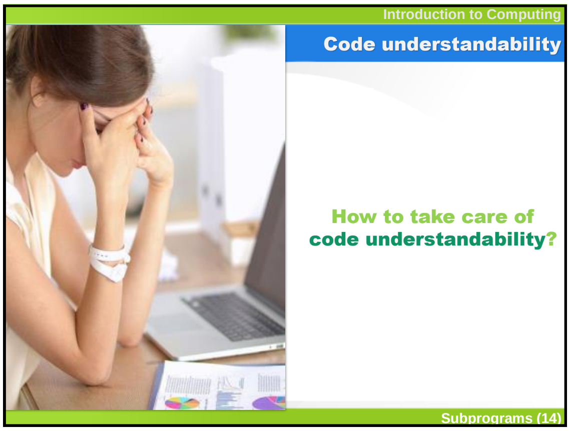


Are those programs correct?

```
# H = heads of cows & hens
# L = legs  of cows & hens
# Given: H and L.
# Find: the number of hens.


H = int(input())
L = int(input()
r = 2*H - L//2
print(r)
```

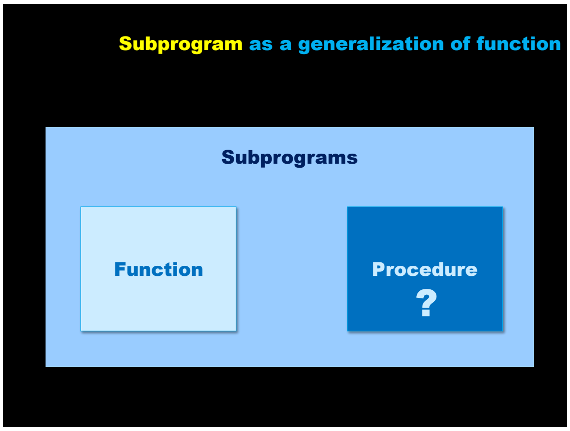
```
/* H = heads of cows & hens
   L = legs  of cows & hens
   Given: H and L.
   Find: the number of hens.
*/
#include <stdio.h>
int main(){
    int H, L, r;
    scanf("%d", &H);
    scanf("%d", &L);
    r = 2*H - L/2;
    printf("%d\n", r);
    }
```

C



Code understandability

How to take care of code understandability?

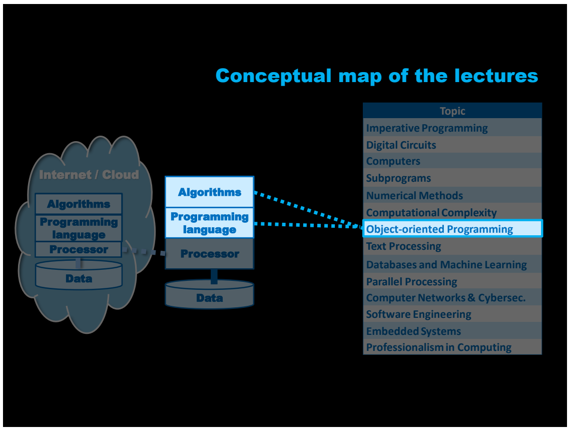


Subprogram as a generalization of function

Subprograms

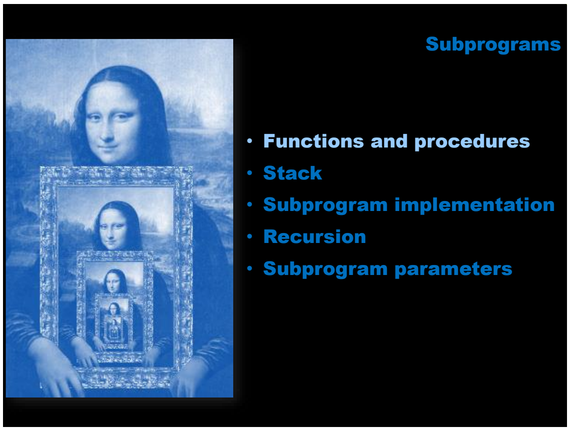
Function

Procedure ?



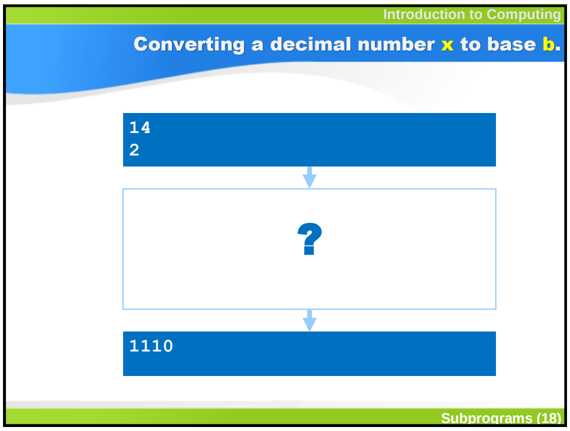
Conceptual map of the lectures

Topic
Imperative Programming
Digital Circuits
Computers
Subprograms
Numerical Methods
Computational Complexity
Object-oriented Programming
Text Processing
Databases and Machine Learning
Parallel Processing
Computer Networks & Cybersec.
Software Engineering
Embedded Systems
Professionalism in Computing



Subprograms

- Functions and procedures
- Stack
- Subprogram implementation
- Recursion
- Subprogram parameters

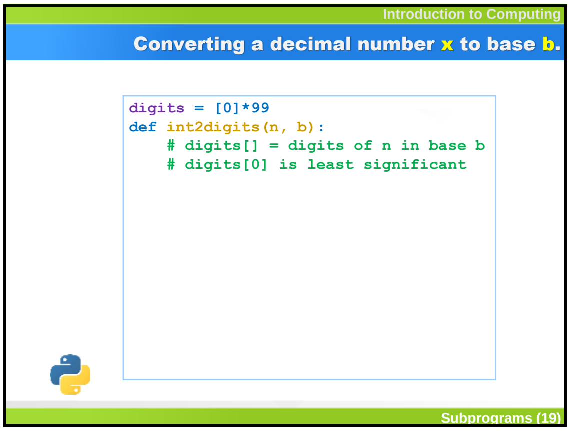


Converting a decimal number x to base b.

14
2

?

1110

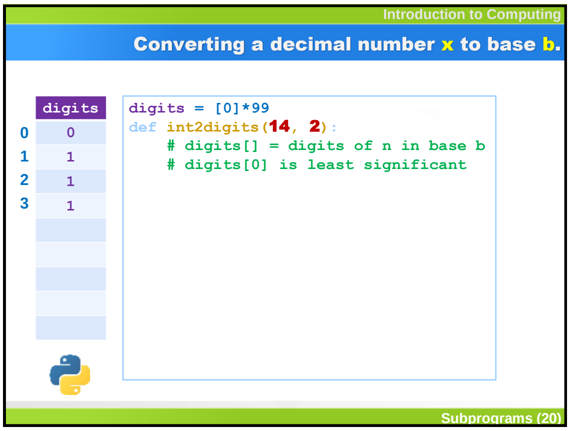


Introduction to Computing

Converting a decimal number x to base b.

```
digits = [0]*99
def int2digits(n, b):
    # digits[] = digits of n in base b
    # digits[0] is least significant
```

Subprograms (19)



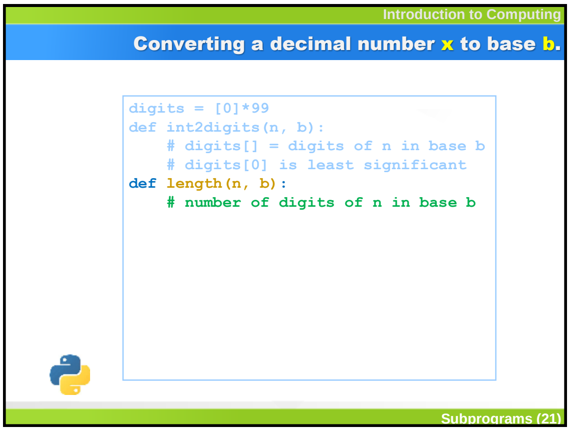
Introduction to Computing

Converting a decimal number x to base b.

	digits
0	0
1	1
2	1
3	1

```
digits = [0]*99
def int2digits(14, 2):
    # digits[] = digits of n in base b
    # digits[0] is least significant
```

Subprograms (20)

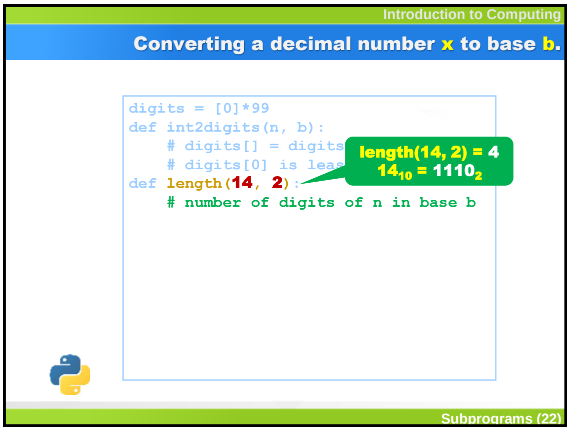


Introduction to Computing

Converting a decimal number x to base b.

```
digits = [0]*99
def int2digits(n, b):
    # digits[] = digits of n in base b
    # digits[0] is least significant
def length(n, b):
    # number of digits of n in base b
```

Subprograms (21)



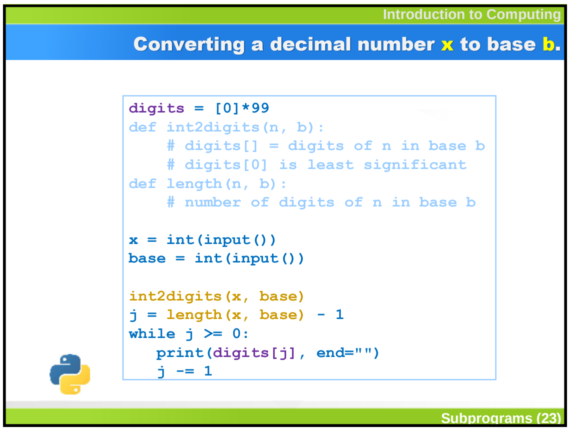
Introduction to Computing

Converting a decimal number x to base b.

```
digits = [0]*99
def int2digits(n, b):
    # digits[] = digits
    # digits[0] is leas
def length(14, 2):
    # number of digits of n in base b
```

length(14, 2) = 4
$14_{10} = 1110_2$

Subprograms (22)

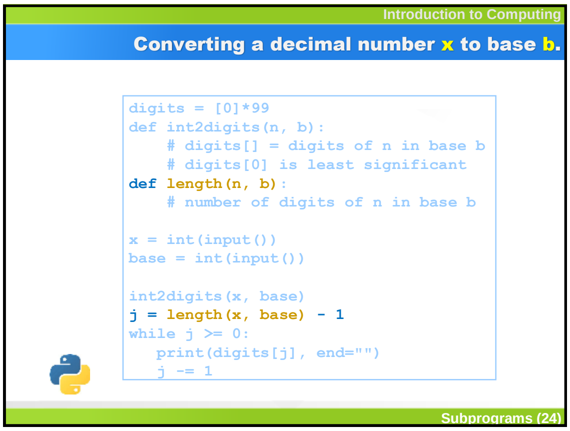


Introduction to Computing

Converting a decimal number x to base b.

```
digits = [0]*99
def int2digits(n, b):
    # digits[] = digits of n in base b
    # digits[0] is least significant
def length(n, b):
    # number of digits of n in base b

x = int(input())
base = int(input())

int2digits(x, base)
j = length(x, base) - 1
while j >= 0:
   print(digits[j], end="")
   j -= 1
```

Subprograms (23)

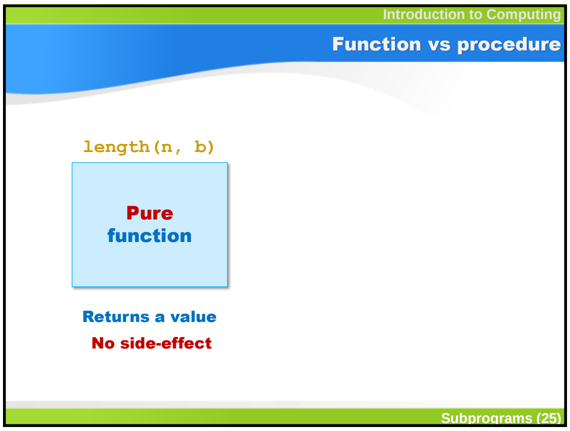


Introduction to Computing

Converting a decimal number x to base b.

```
digits = [0]*99
def int2digits(n, b):
    # digits[] = digits of n in base b
    # digits[0] is least significant
def length(n, b):
    # number of digits of n in base b

x = int(input())
base = int(input())

int2digits(x, base)
j = length(x, base) - 1
while j >= 0:
   print(digits[j], end="")
   j -= 1
```

Subprograms (24)



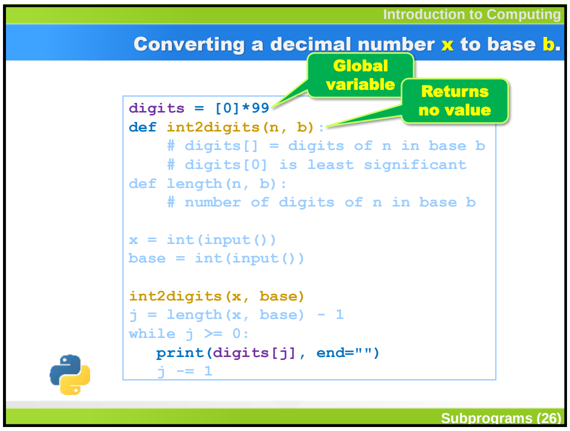
Function vs procedure

length(n, b)

Pure function

Returns a value

No side-effect

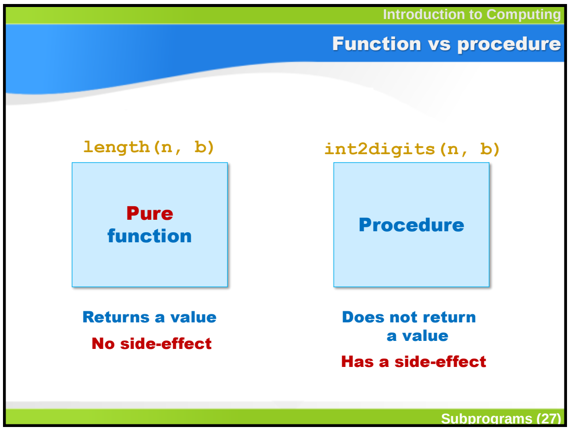


Converting a decimal number x to base b.

```
digits = [0]*99
def int2digits(n, b):
    # digits[] = digits of n in base b
    # digits[0] is least significant
def length(n, b):
    # number of digits of n in base b

x = int(input())
base = int(input())

int2digits(x, base)
j = length(x, base) - 1
while j >= 0:
   print(digits[j], end="")
   j -= 1
```

Global variable

Returns no value



Function vs procedure

length(n, b)

Pure function

Returns a value

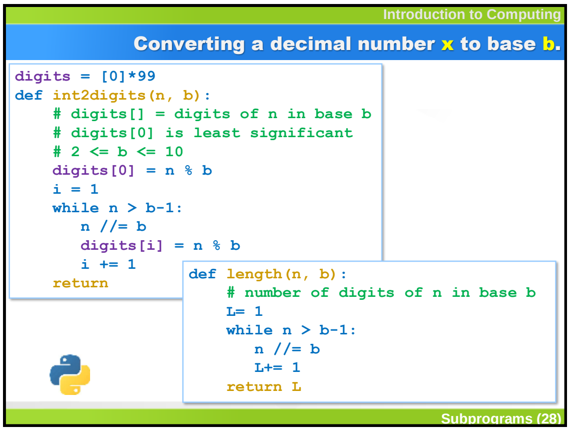
No side-effect

int2digits(n, b)

Procedure

Does not return a value

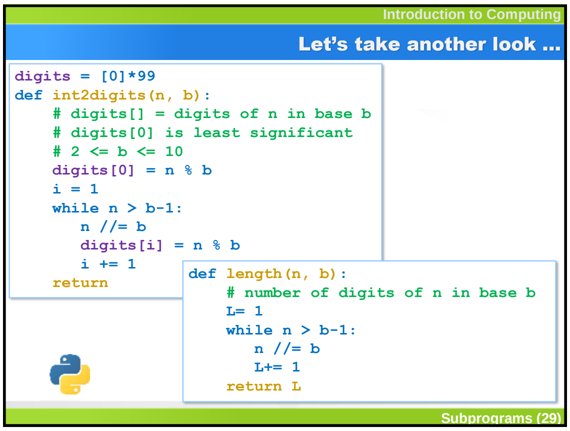
Has a side-effect



Converting a decimal number x to base b.

```
digits = [0]*99
def int2digits(n, b):
    # digits[] = digits of n in base b
    # digits[0] is least significant
    # 2 <= b <= 10
    digits[0] = n % b
    i = 1
    while n > b-1:
       n //= b
       digits[i] = n % b
       i += 1
    return
```

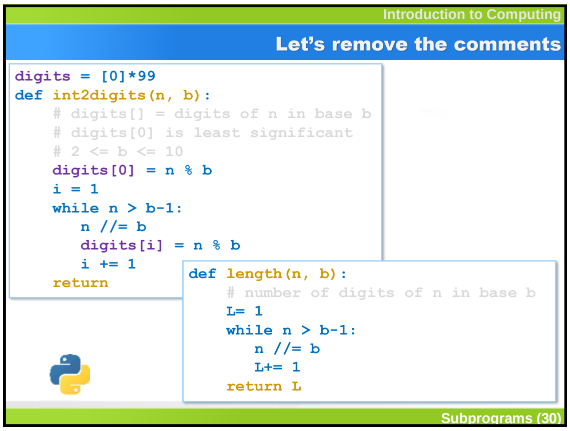
```
def length(n, b):
    # number of digits of n in base b
    L= 1
    while n > b-1:
       n //= b
       L+= 1
    return L
```



Let’s take another look …

```
digits = [0]*99
def int2digits(n, b):
    # digits[] = digits of n in base b
    # digits[0] is least significant
    # 2 <= b <= 10
    digits[0] = n % b
    i = 1
    while n > b-1:
       n //= b
       digits[i] = n % b
       i += 1
    return
```

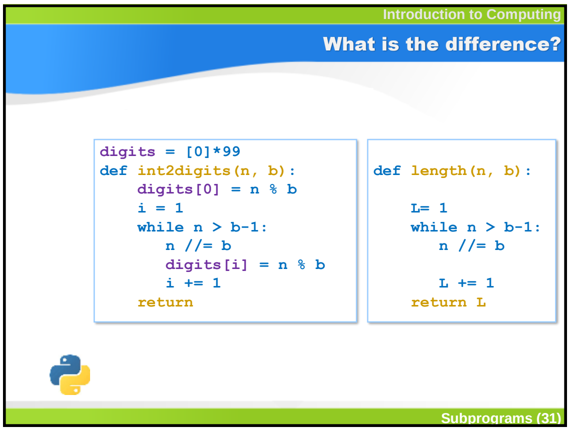
```
def length(n, b):
    # number of digits of n in base b
    L= 1
    while n > b-1:
       n //= b
       L+= 1
    return L
```



Let’s remove the comments

```
digits = [0]*99
def int2digits(n, b):
    # digits[] = digits of n in base b
    # digits[0] is least significant
    # 2 <= b <= 10
    digits[0] = n % b
    i = 1
    while n > b-1:
       n //= b
       digits[i] = n % b
       i += 1
    return
```

```
def length(n, b):
    # number of digits of n in base b
    L= 1
    while n > b-1:
       n //= b
       L+= 1
    return L
```



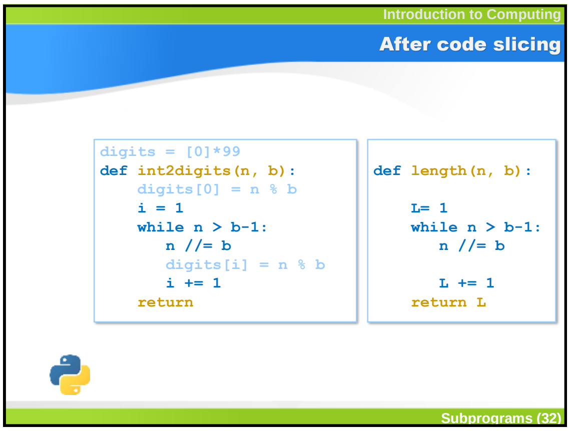
Introduction to Computing

What is the difference?

```
digits = [0]*99
def int2digits(n, b):
    digits[0] = n % b
    i = 1
    while n > b-1:
       n //= b
       digits[i] = n % b
       i += 1
    return
```

```
def length(n, b):

    L= 1
    while n > b-1:
       n //= b

       L += 1
    return L
```

Subprograms (31)



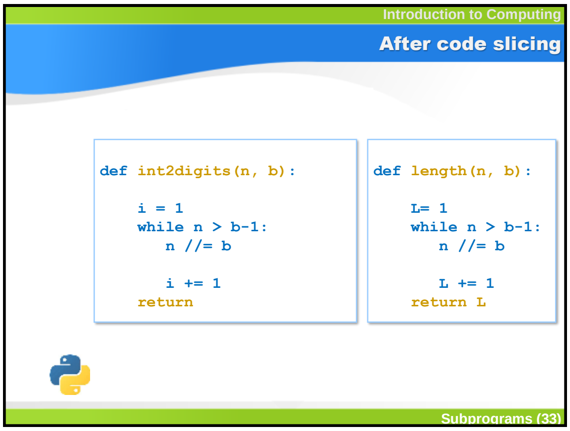
Introduction to Computing

After code slicing

```
digits = [0]*99
def int2digits(n, b):
    digits[0] = n % b
    i = 1
    while n > b-1:
       n //= b
       digits[i] = n % b
       i += 1
    return
```

```
def length(n, b):

    L= 1
    while n > b-1:
       n //= b

       L += 1
    return L
```

Subprograms (32)



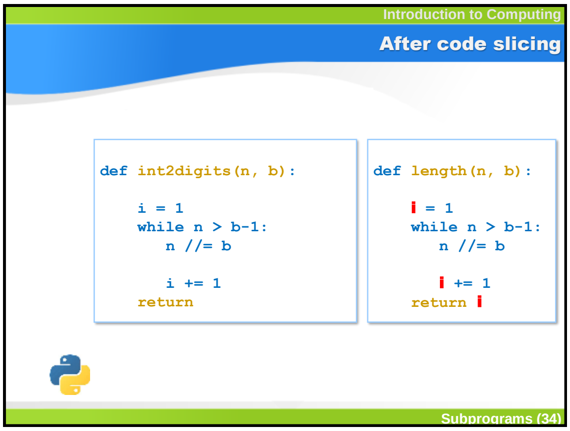
Introduction to Computing

After code slicing

```
def int2digits(n, b):

    i = 1
    while n > b-1:
       n //= b

       i += 1
    return
```

```
def length(n, b):

    L= 1
    while n > b-1:
       n //= b

       L += 1
    return L
```

Subprograms (33)



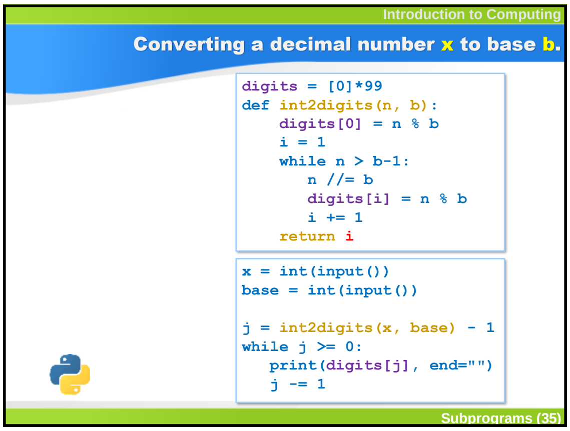
Introduction to Computing

After code slicing

```
def int2digits(n, b):

    i = 1
    while n > b-1:
       n //= b

       i += 1
    return
```

```
def length(n, b):

    i = 1
    while n > b-1:
       n //= b

       i += 1
    return i
```

Subprograms (34)



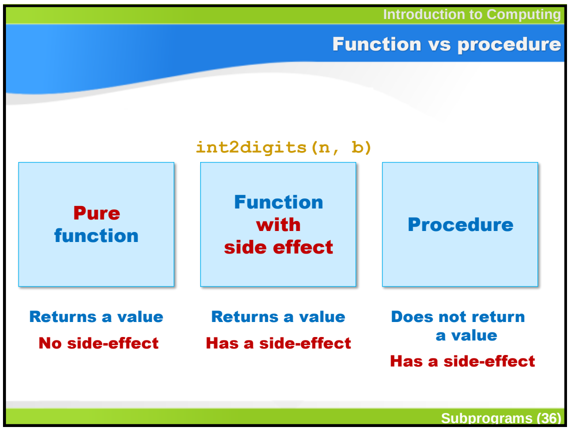
Introduction to Computing

Converting a decimal number x to base b.

```
digits = [0]*99
def int2digits(n, b):
    digits[0] = n % b
    i = 1
    while n > b-1:
       n //= b
       digits[i] = n % b
       i += 1
    return i
```

```
x = int(input())
base = int(input())

j = int2digits(x, base) - 1
while j >= 0:
   print(digits[j], end="")
   j -= 1
```

Subprograms (35)



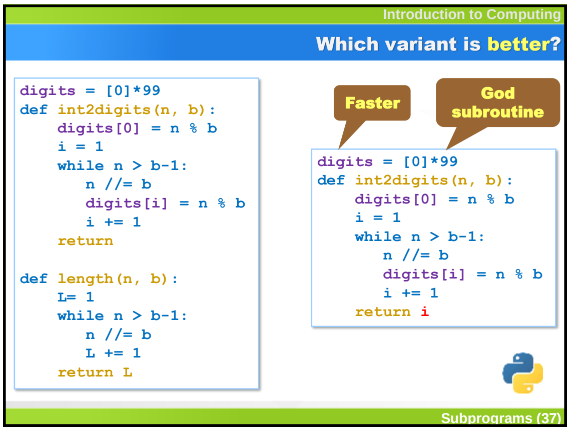
Introduction to Computing

Function vs procedure

`int2digits(n, b)`

Pure function	Function with side effect	Procedure
Returns a value	Returns a value	Does not return a value
No side-effect	Has a side-effect	Has a side-effect

Subprograms (36)



Introduction to Computing

Which variant is better?

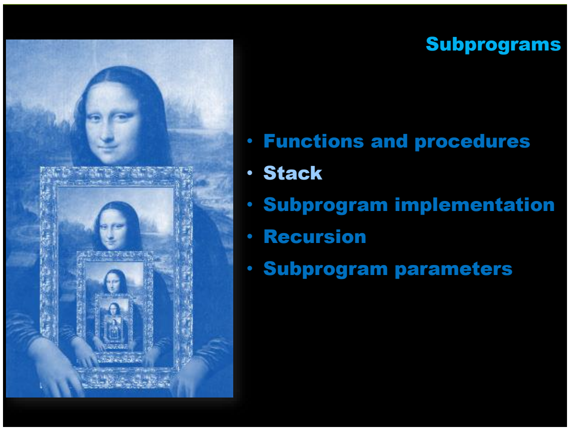
```
digits = [0]*99
def int2digits(n, b):
    digits[0] = n % b
    i = 1
    while n > b-1:
        n //= b
        digits[i] = n % b
        i += 1
    return

def length(n, b):
    L= 1
    while n > b-1:
        n //= b
        L += 1
    return L
```

Faster

God subroutine

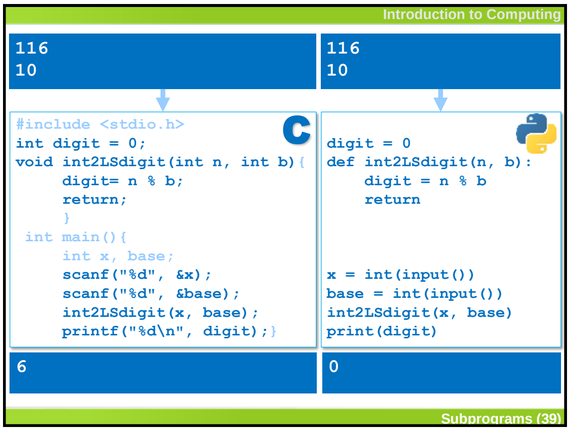
```
digits = [0]*99
def int2digits(n, b):
    digits[0] = n % b
    i = 1
    while n > b-1:
        n //= b
        digits[i] = n % b
        i += 1
    return i
```

Subprograms (37)



Subprograms

- Functions and procedures
- Stack
- Subprogram implementation
- Recursion
- Subprogram parameters



Introduction to Computing

```
116
10
```

```
#include <stdio.h>
int digit = 0;
void int2LSdigit(int n, int b){
     digit= n % b;
     return;
     }
 int main(){
     int x, base;
     scanf("%d", &x);
     scanf("%d", &base);
     int2LSdigit(x, base);
     printf("%d\n", digit);}
```

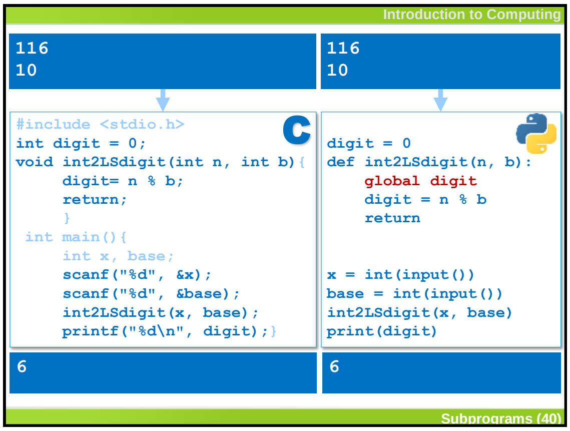
```
6
```

```
116
10
```

```
digit = 0
def int2LSdigit(n, b):
    digit = n % b
    return


x = int(input())
base = int(input())
int2LSdigit(x, base)
print(digit)
```

```
0
```

Subprograms (39)



Introduction to Computing

```
116
10
```

```
#include <stdio.h>
int digit = 0;
void int2LSdigit(int n, int b){
     digit= n % b;
     return;
     }
 int main(){
     int x, base;
     scanf("%d", &x);
     scanf("%d", &base);
     int2LSdigit(x, base);
     printf("%d\n", digit);}
```

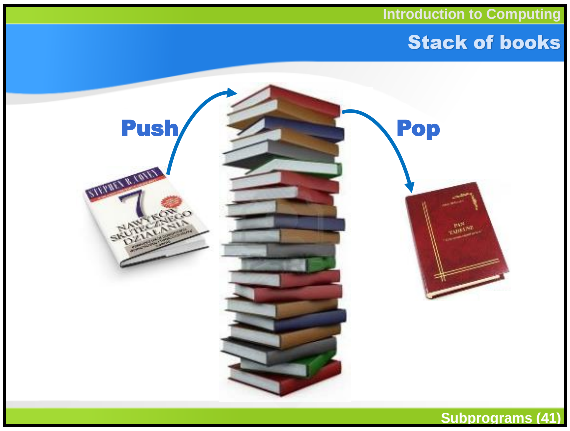
```
6
```

```
116
10
```

```
digit = 0
def int2LSdigit(n, b):
    global digit
    digit = n % b
    return

x = int(input())
base = int(input())
int2LSdigit(x, base)
print(digit)
```

```
6
```

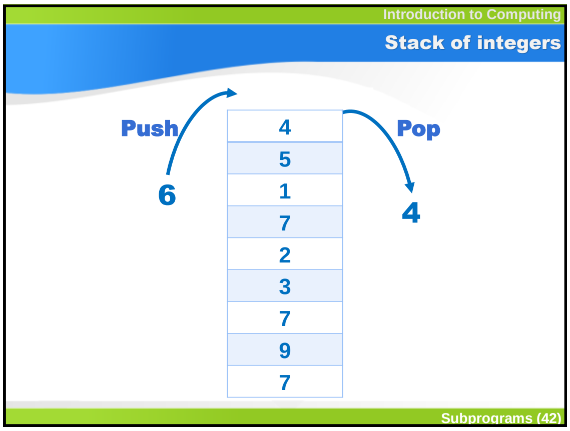
Subprograms (40)



Introduction to Computing

Stack of books

Subprograms (41)

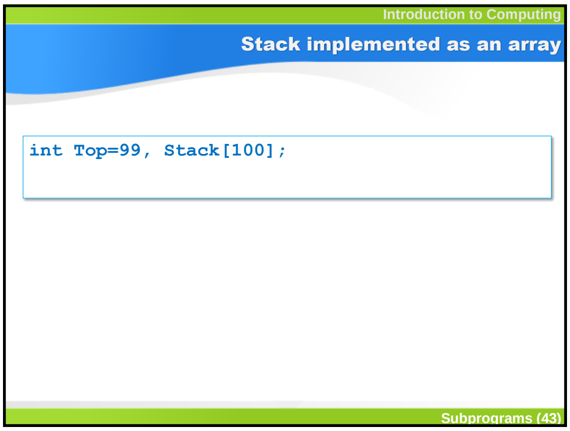


Introduction to Computing

Stack of integers

4
5
1
7
2
3
7
9
7

Subprograms (42)

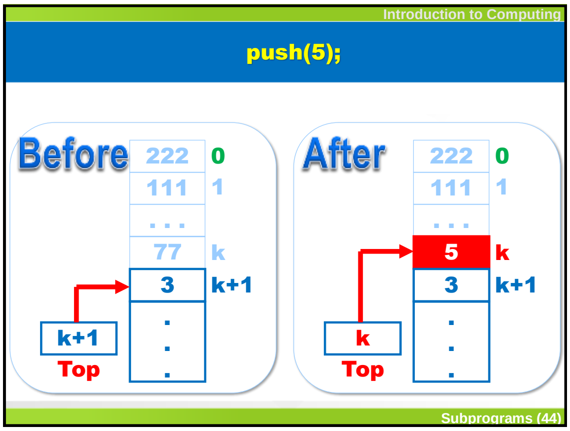


Introduction to Computing

Stack implemented as an array

```
int Top=99, Stack[100];
```

Subprograms (43)



Introduction to Computing

push(5);

Before

222	0
111	1
. . .	
77	k
3	k+1
: :	

k+1

Top

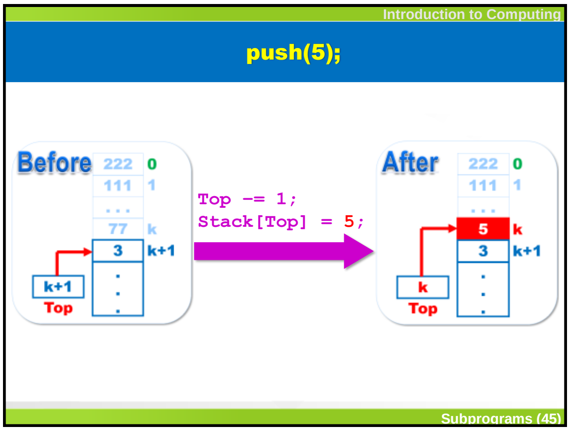
After

222	0
111	1
. . .	
5	k
3	k+1
: :	

k

Top

Subprograms (44)



Introduction to Computing

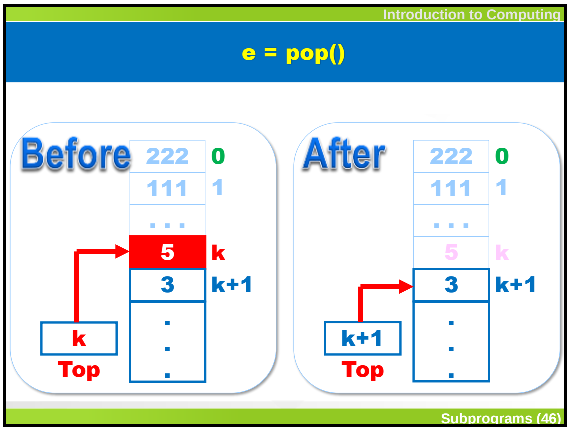
push(5);

Before: 222 0 / 111 1 / . . . / 77 k / 3 k+1; Top = k+1

```
Top -= 1;
Stack[Top] = 5;
```

After: 222 0 / 111 1 / . . . / 5 k / 3 k+1; Top = k

Subprograms (45)



Introduction to Computing

e = pop()

Before

222	0
111	1
. . .	
5	k
3	k+1
: :	

k

Top

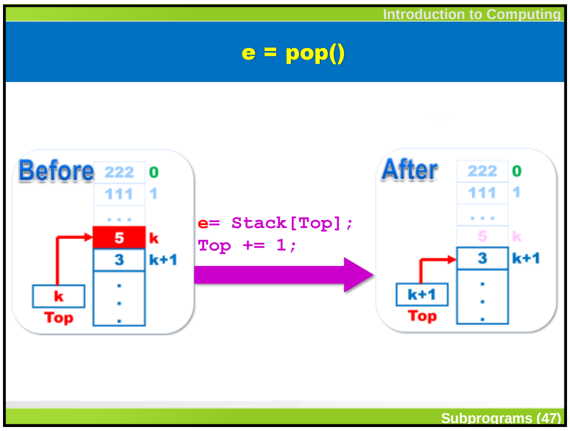
After

222	0
111	1
. . .	
5	k
3	k+1
: :	

k+1

Top

Subprograms (46)



Introduction to Computing

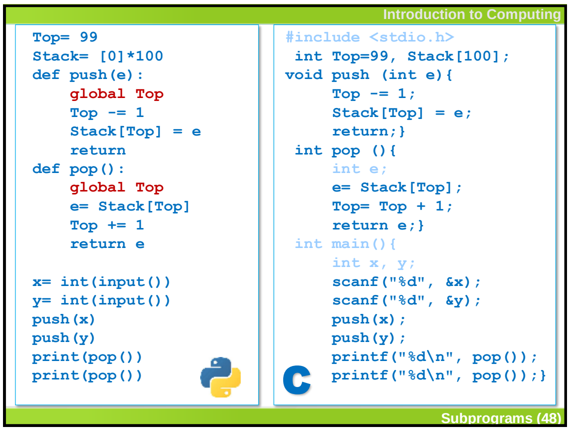
e = pop()

Before: 222 0 / 111 1 / . . . / 5 k / 3 k+1; Top = k

```
e= Stack[Top];
Top += 1;
```

After: 222 0 / 111 1 / . . . / 5 k / 3 k+1; Top = k+1

Subprograms (47)

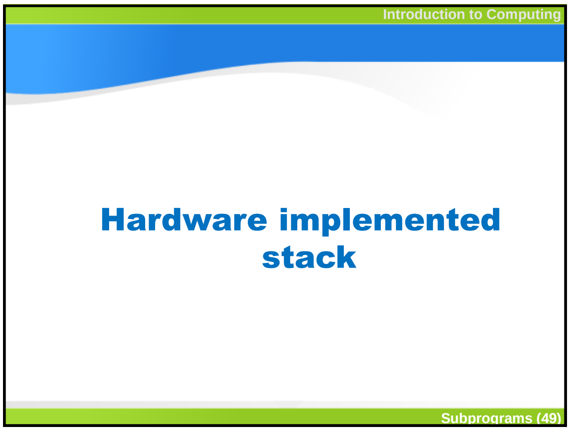


Introduction to Computing

```
Top= 99
Stack= [0]*100
def push(e):
    global Top
    Top -= 1
    Stack[Top] = e
    return
def pop():
    global Top
    e= Stack[Top]
    Top += 1
    return e

x= int(input())
y= int(input())
push(x)
push(y)
print(pop())
print(pop())
```

```
#include <stdio.h>
 int Top=99, Stack[100];
void push (int e){
     Top -= 1;
     Stack[Top] = e;
     return;}
 int pop (){
     int e;
     e= Stack[Top];
     Top= Top + 1;
     return e;}
 int main(){
     int x, y;
     scanf("%d", &x);
     scanf("%d", &y);
     push(x);
     push(y);
     printf("%d\n", pop());
     printf("%d\n", pop());}
```

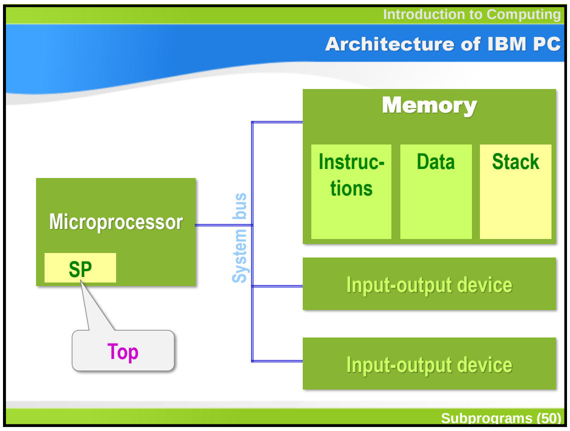
Subprograms (48)



Introduction to Computing

Hardware implemented stack

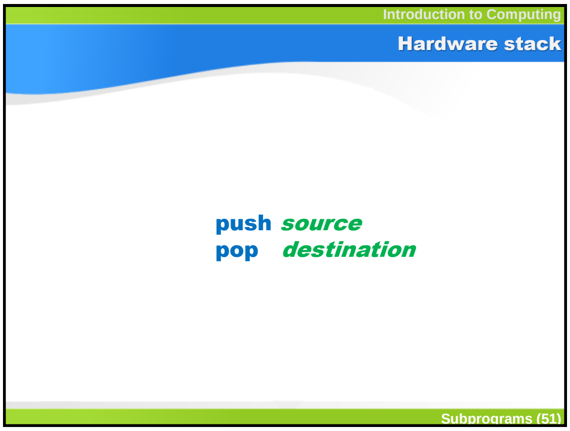
Subprograms (49)



Introduction to Computing

Architecture of IBM PC

Subprograms (50)

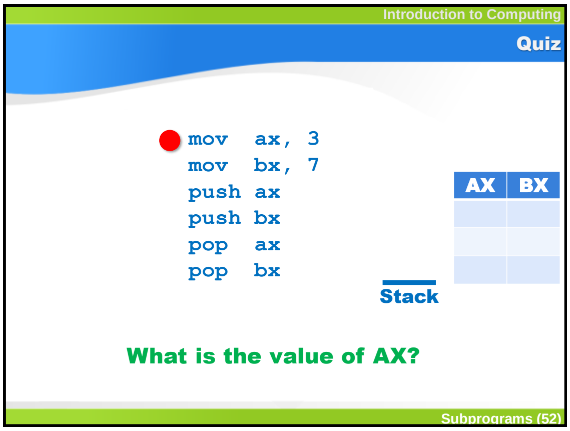


Introduction to Computing

Hardware stack

push ***source***
pop ***destination***

Subprograms (51)



Introduction to Computing

Quiz

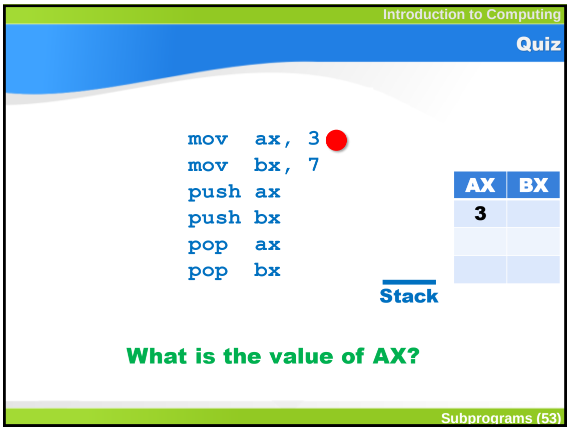
```
● mov   ax, 3
  mov   bx, 7
  push  ax
  push  bx
  pop   ax
  pop   bx
```

AX	BX

Stack

What is the value of AX?

Subprograms (52)



Introduction to Computing

Quiz

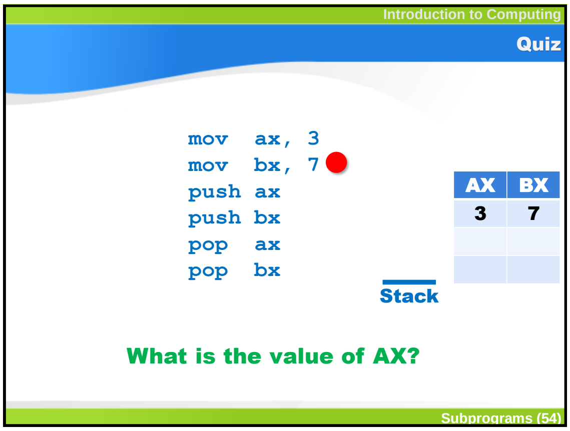
```
mov   ax, 3 ●
mov   bx, 7
push  ax
push  bx
pop   ax
pop   bx
```

AX	BX
3	

Stack

What is the value of AX?

Subprograms (53)



Introduction to Computing

Quiz

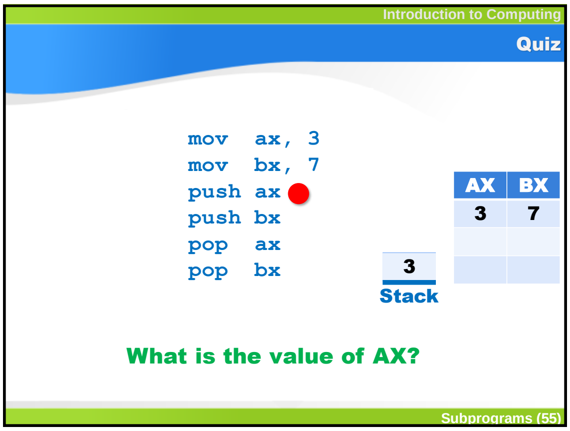
```
mov   ax, 3
mov   bx, 7 ●
push  ax
push  bx
pop   ax
pop   bx
```

AX	BX
3	7

Stack

What is the value of AX?

Subprograms (54)



Introduction to Computing

Quiz

```
mov   ax, 3
mov   bx, 7
push  ax ●
push  bx
pop   ax
pop   bx
```

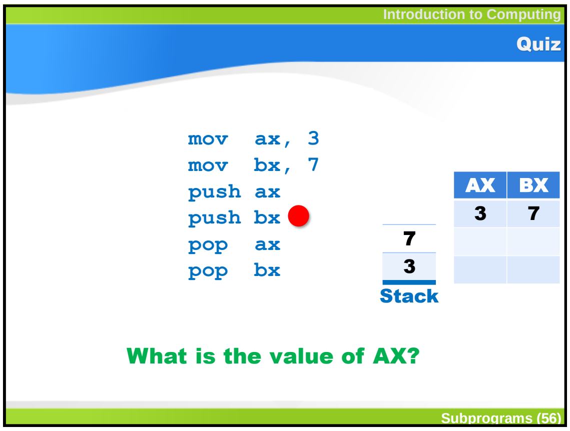
AX	BX
3	7

3

Stack

What is the value of AX?

Subprograms (55)



Introduction to Computing

Quiz

```
mov   ax, 3
mov   bx, 7
push  ax
push  bx ●
pop   ax
pop   bx
```

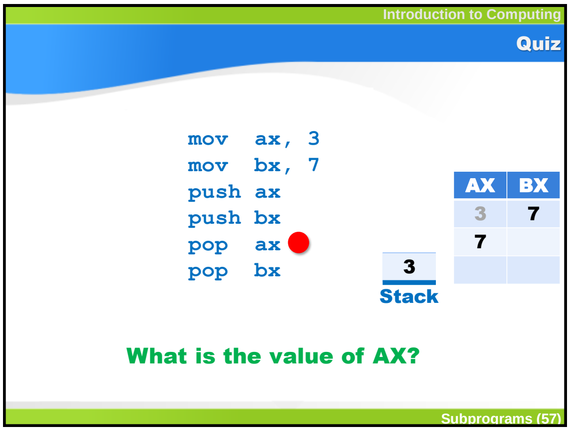
AX	BX
3	7

7
3

Stack

What is the value of AX?

Subprograms (56)



Introduction to Computing

Quiz

```
mov   ax, 3
mov   bx, 7
push  ax
push  bx
pop   ax ●
pop   bx
```

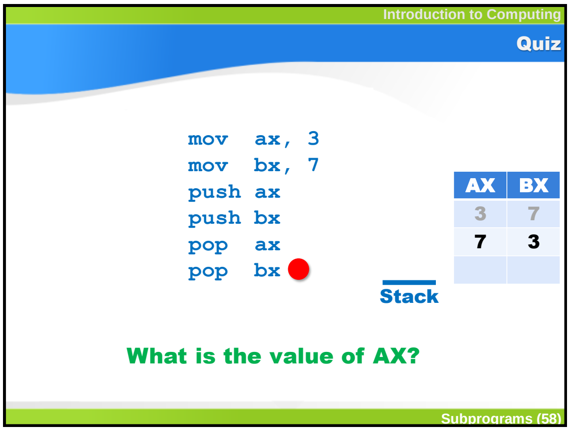
AX	BX
3	7
7	

3

Stack

What is the value of AX?

Subprograms (57)



Introduction to Computing

Quiz

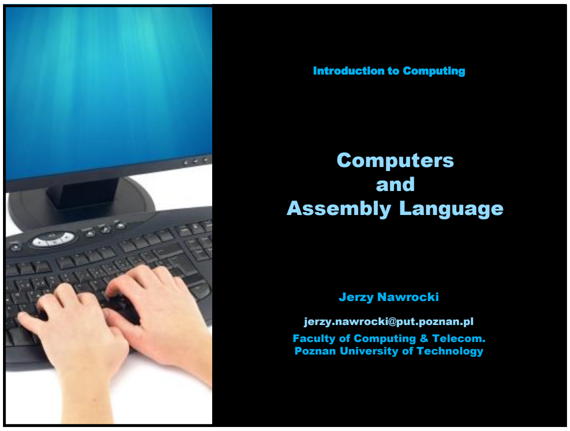
```
mov   ax, 3
mov   bx, 7
push  ax
push  bx
pop   ax
pop   bx ●
```

AX	BX
3	7
7	3

Stack

What is the value of AX?

Subprograms (58)

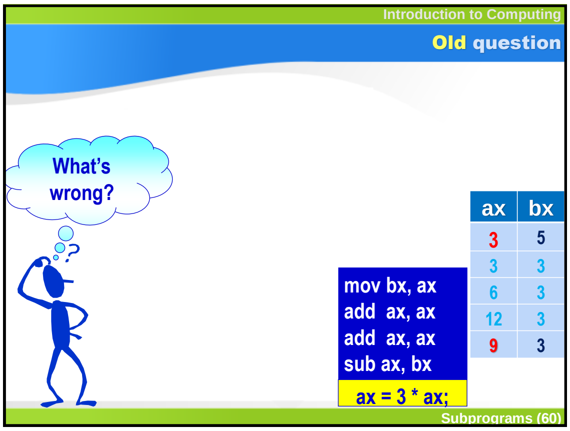


Introduction to Computing

Computers
and
Assembly Language

Jerzy Nawrocki

jerzy.nawrocki@put.poznan.pl
Faculty of Computing & Telecom.
Poznan University of Technology



Introduction to Computing

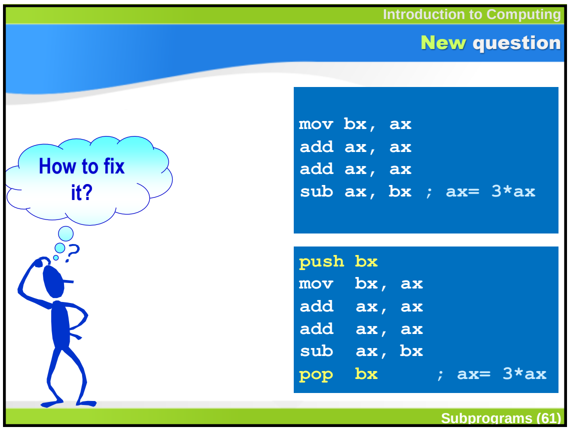
Old question

```
mov bx, ax
add  ax, ax
add  ax, ax
sub ax, bx
```

ax = 3 * ax;

ax	bx
3	5
3	3
6	3
12	3
9	3

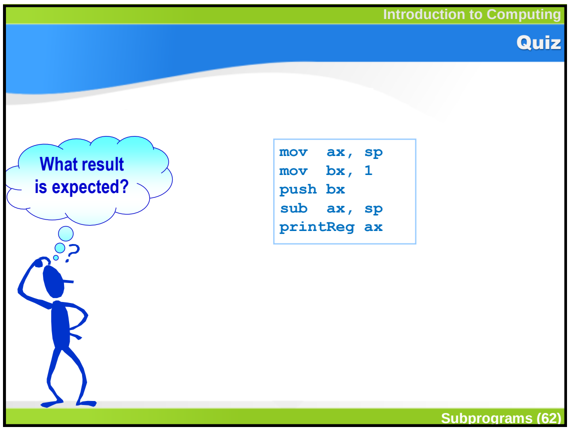
Subprograms (60)



New question

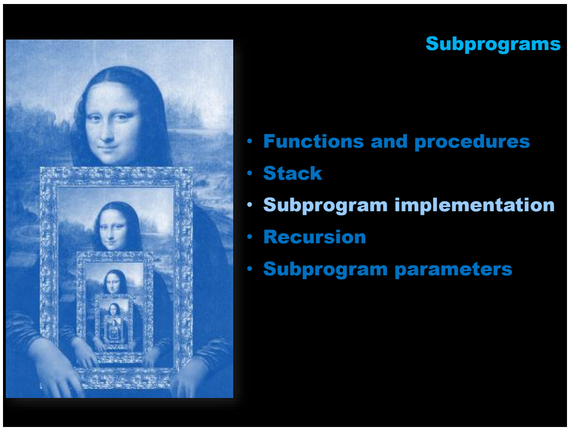
```
mov bx, ax
add ax, ax
add ax, ax
sub ax, bx ; ax= 3*ax
```

```
push bx
mov  bx, ax
add  ax, ax
add  ax, ax
sub  ax, bx
pop  bx      ; ax= 3*ax
```



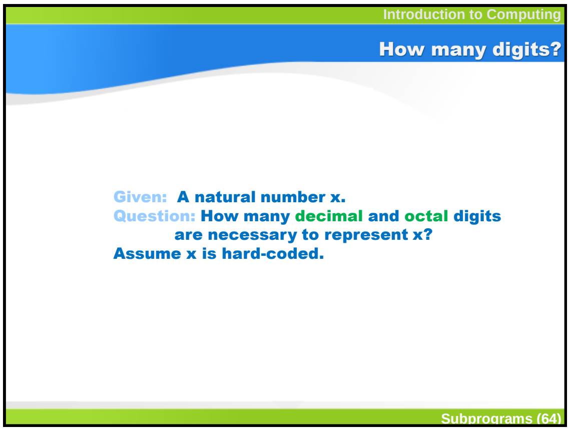
Quiz

```
mov  ax, sp
mov  bx, 1
push bx
sub  ax, sp
printReg ax
```



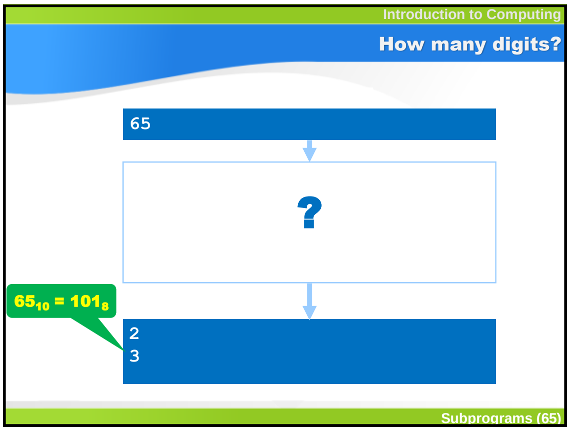
Subprograms

- Functions and procedures
- Stack
- Subprogram implementation
- Recursion
- Subprogram parameters



How many digits?

Given: A natural number x.
Question: How many decimal and octal digits are necessary to represent x?
Assume x is hard-coded.



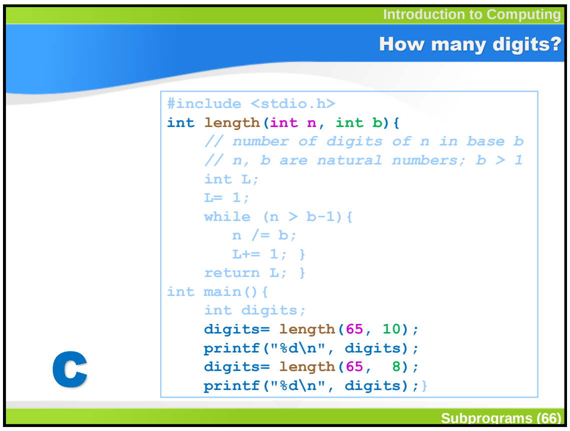
How many digits?

65

?

2
3

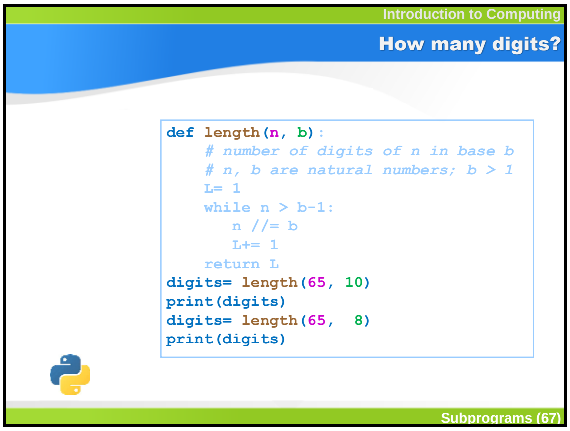
$65_{10} = 101_8$



How many digits?

C

```
#include <stdio.h>
int length(int n, int b){
    // number of digits of n in base b
    // n, b are natural numbers; b > 1
    int L;
    L= 1;
    while (n > b-1){
       n /= b;
       L+= 1; }
    return L; }
int main(){
    int digits;
    digits= length(65, 10);
    printf("%d\n", digits);
    digits= length(65,  8);
    printf("%d\n", digits);}
```

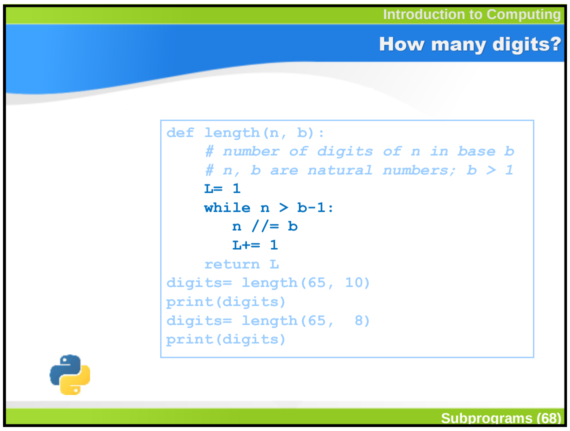


Introduction to Computing

How many digits?

```
def length(n, b):
    # number of digits of n in base b
    # n, b are natural numbers; b > 1
    L= 1
    while n > b-1:
       n //= b
       L+= 1
    return L
digits= length(65, 10)
print(digits)
digits= length(65,  8)
print(digits)
```

Subprograms (67)

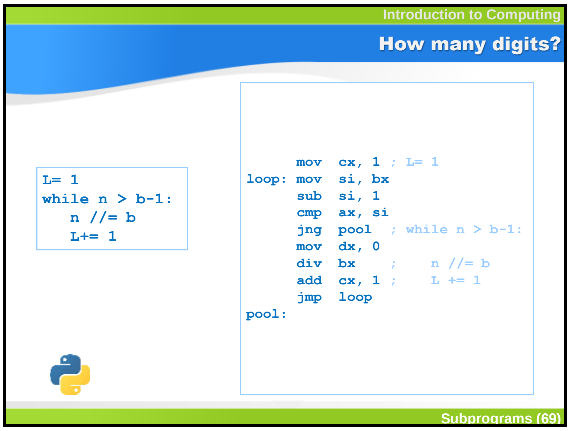


Introduction to Computing

How many digits?

```
def length(n, b):
    # number of digits of n in base b
    # n, b are natural numbers; b > 1
    L= 1
    while n > b-1:
       n //= b
       L+= 1
    return L
digits= length(65, 10)
print(digits)
digits= length(65,  8)
print(digits)
```

Subprograms (68)



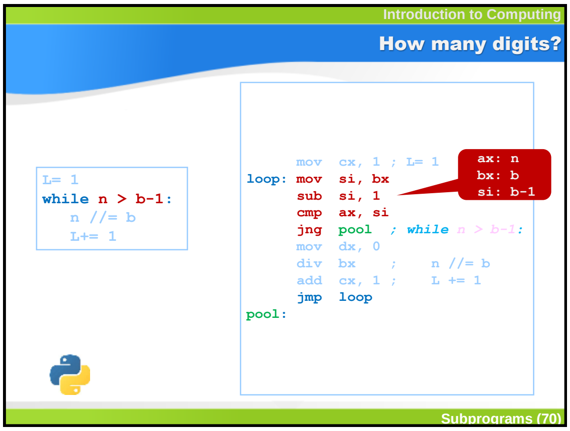
Introduction to Computing

How many digits?

```
L= 1
while n > b-1:
   n //= b
   L+= 1
```

```
      mov  cx, 1 ; L= 1
loop: mov  si, bx
      sub  si, 1
      cmp  ax, si
      jng  pool   ; while n > b-1:
      mov  dx, 0
      div  bx     ;    n //= b
      add  cx, 1 ;    L += 1
      jmp  loop
pool:
```

Subprograms (69)



Introduction to Computing

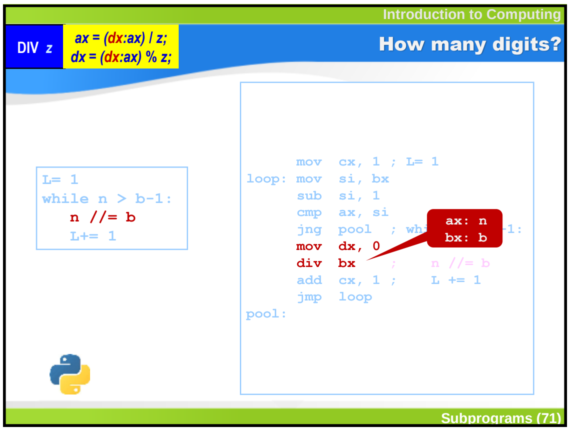
How many digits?

```
L= 1
while n > b-1:
   n //= b
   L+= 1
```

```
      mov  cx, 1 ; L= 1
loop: mov  si, bx
      sub  si, 1
      cmp  ax, si
      jng  pool   ; while n > b-1:
      mov  dx, 0
      div  bx     ;    n //= b
      add  cx, 1 ;    L += 1
      jmp  loop
pool:
```

ax: n
bx: b
si: b-1

Subprograms (70)



DIV *z*

ax = (dx:ax) / z;
dx = (dx:ax) % z;

Introduction to Computing

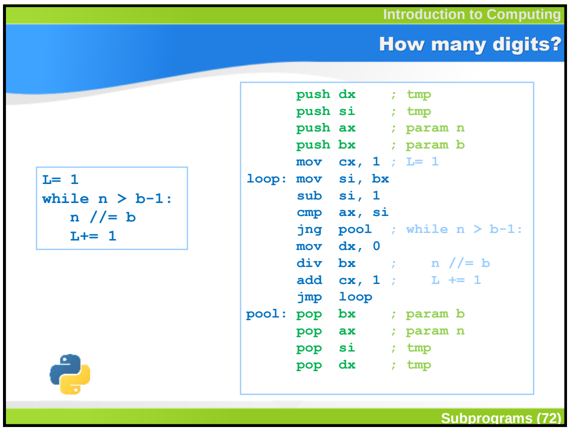
How many digits?

```
L= 1
while n > b-1:
   n //= b
   L+= 1
```

```
      mov  cx, 1 ; L= 1
loop: mov  si, bx
      sub  si, 1
      cmp  ax, si
      jng  pool   ; wh          -1:
      mov  dx, 0
      div  bx     ;    n //= b
      add  cx, 1 ;    L += 1
      jmp  loop
pool:
```

ax: n
bx: b

Subprograms (71)



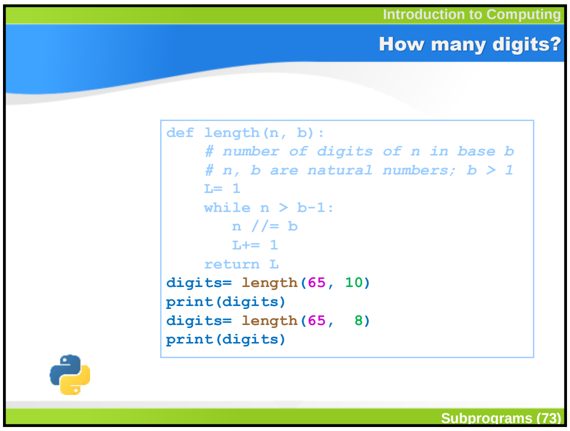
Introduction to Computing

How many digits?

```
L= 1
while n > b-1:
   n //= b
   L+= 1
```

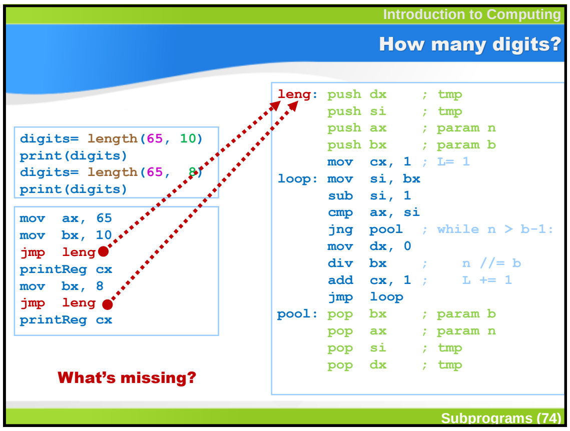
```
      push dx    ; tmp
      push si    ; tmp
      push ax    ; param n
      push bx    ; param b
      mov  cx, 1 ; L= 1
loop: mov  si, bx
      sub  si, 1
      cmp  ax, si
      jng  pool   ; while n > b-1:
      mov  dx, 0
      div  bx     ;    n //= b
      add  cx, 1 ;    L += 1
      jmp  loop
pool: pop  bx    ; param b
      pop  ax    ; param n
      pop  si    ; tmp
      pop  dx    ; tmp
```

Subprograms (72)



How many digits?

```
def length(n, b):
    # number of digits of n in base b
    # n, b are natural numbers; b > 1
    L= 1
    while n > b-1:
       n //= b
       L+= 1
    return L
digits= length(65, 10)
print(digits)
digits= length(65,  8)
print(digits)
```



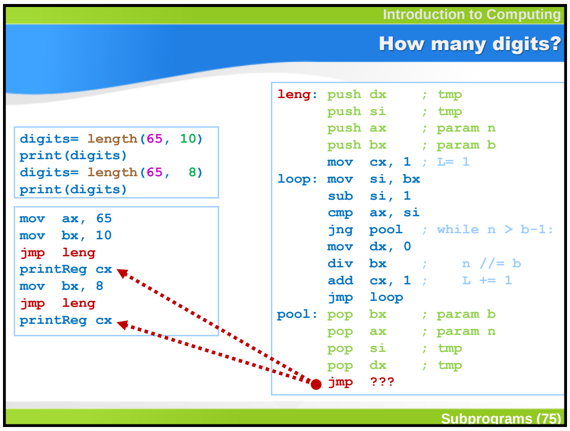
How many digits?

```
digits= length(65, 10)
print(digits)
digits= length(65,  8)
print(digits)
```

```
mov  ax, 65
mov  bx, 10
jmp  leng
printReg cx
mov  bx, 8
jmp  leng
printReg cx
```

What's missing?

```
leng: push dx     ; tmp
      push si     ; tmp
      push ax     ; param n
      push bx     ; param b
      mov  cx, 1 ; L= 1
loop: mov  si, bx
      sub  si, 1
      cmp  ax, si
      jng  pool   ; while n > b-1:
      mov  dx, 0
      div  bx     ;     n //= b
      add  cx, 1 ;      L += 1
      jmp  loop
pool: pop  bx     ; param b
      pop  ax     ; param n
      pop  si     ; tmp
      pop  dx     ; tmp
```

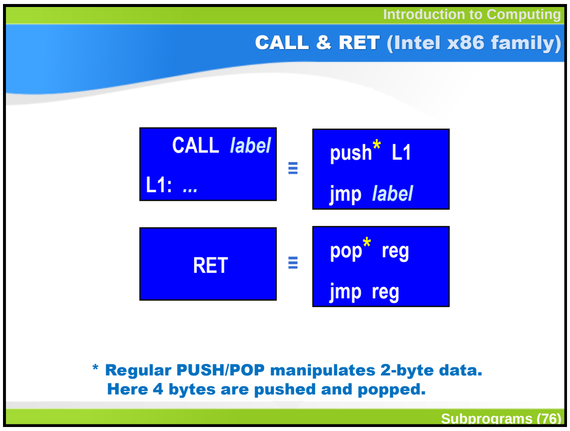


How many digits?

```
digits= length(65, 10)
print(digits)
digits= length(65,  8)
print(digits)
```

```
mov  ax, 65
mov  bx, 10
jmp  leng
printReg cx
mov  bx, 8
jmp  leng
printReg cx
```

```
leng: push dx     ; tmp
      push si     ; tmp
      push ax     ; param n
      push bx     ; param b
      mov  cx, 1 ; L= 1
loop: mov  si, bx
      sub  si, 1
      cmp  ax, si
      jng  pool   ; while n > b-1:
      mov  dx, 0
      div  bx     ;     n //= b
      add  cx, 1 ;      L += 1
      jmp  loop
pool: pop  bx     ; param b
      pop  ax     ; param n
      pop  si     ; tmp
      pop  dx     ; tmp
      jmp  ???
```



CALL & RET (Intel x86 family)

CALL *label*
L1: ...

≡

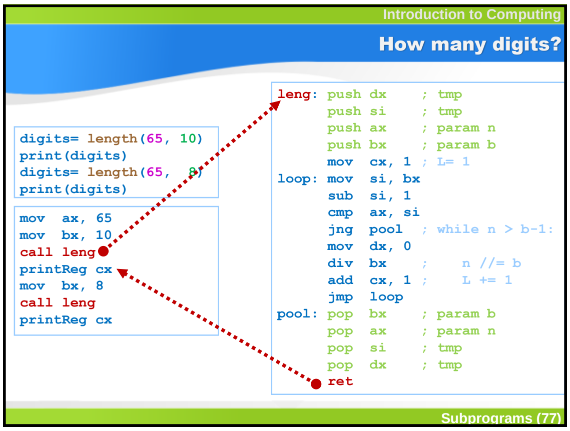
push* L1
jmp *label*

RET

≡

pop* reg
jmp reg

* Regular PUSH/POP manipulates 2-byte data.
Here 4 bytes are pushed and popped.

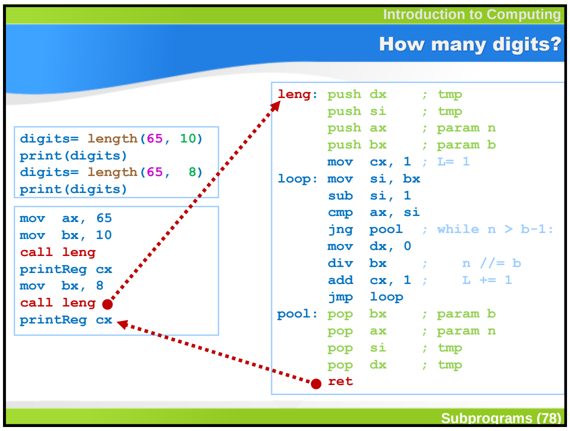


How many digits?

```
digits= length(65, 10)
print(digits)
digits= length(65,  8)
print(digits)
```

```
mov  ax, 65
mov  bx, 10
call leng
printReg cx
mov  bx, 8
call leng
printReg cx
```

```
leng: push dx     ; tmp
      push si     ; tmp
      push ax     ; param n
      push bx     ; param b
      mov  cx, 1 ; L= 1
loop: mov  si, bx
      sub  si, 1
      cmp  ax, si
      jng  pool   ; while n > b-1:
      mov  dx, 0
      div  bx     ;     n //= b
      add  cx, 1 ;      L += 1
      jmp  loop
pool: pop  bx     ; param b
      pop  ax     ; param n
      pop  si     ; tmp
      pop  dx     ; tmp
      ret
```

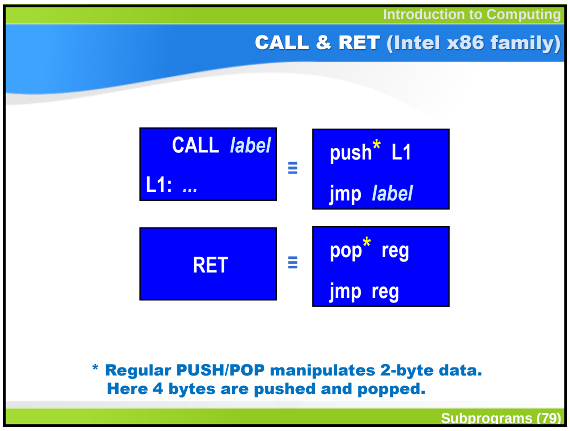


How many digits?

```
digits= length(65, 10)
print(digits)
digits= length(65,  8)
print(digits)
```

```
mov  ax, 65
mov  bx, 10
call leng
printReg cx
mov  bx, 8
call leng
printReg cx
```

```
leng: push dx     ; tmp
      push si     ; tmp
      push ax     ; param n
      push bx     ; param b
      mov  cx, 1 ; L= 1
loop: mov  si, bx
      sub  si, 1
      cmp  ax, si
      jng  pool   ; while n > b-1:
      mov  dx, 0
      div  bx     ;     n //= b
      add  cx, 1 ;      L += 1
      jmp  loop
pool: pop  bx     ; param b
      pop  ax     ; param n
      pop  si     ; tmp
      pop  dx     ; tmp
      ret
```



Introduction to Computing

CALL & RET (Intel x86 family)

CALL *label*
L1: ...

≡

push* L1
jmp *label*

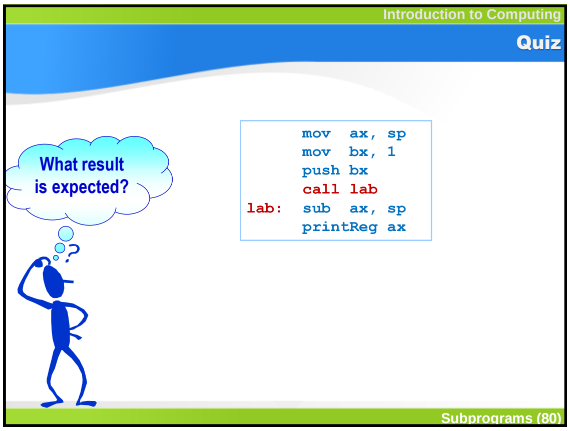
RET

≡

pop* reg
jmp reg

* Regular PUSH/POP manipulates 2-byte data.
Here 4 bytes are pushed and popped.

Subprograms (79)



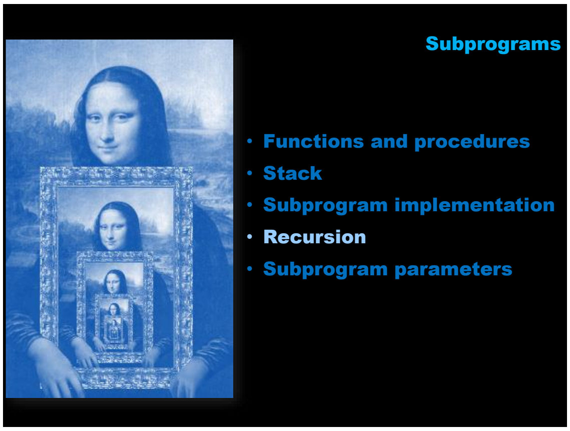
Introduction to Computing

Quiz

What result is expected?

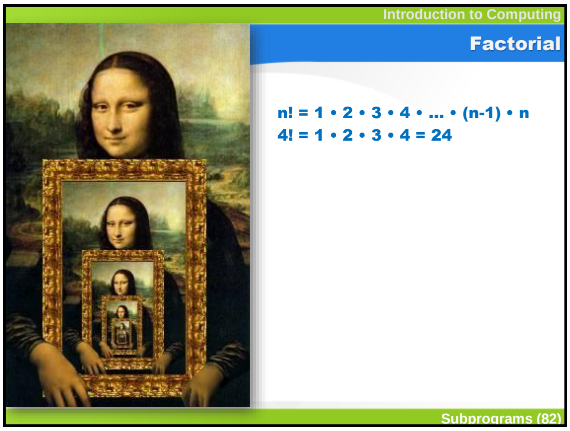
```
      mov  ax, sp
      mov  bx, 1
      push bx
      call lab
lab:  sub  ax, sp
      printReg ax
```

Subprograms (80)



Subprograms

- Functions and procedures
- Stack
- Subprogram implementation
- **Recursion**
- Subprogram parameters

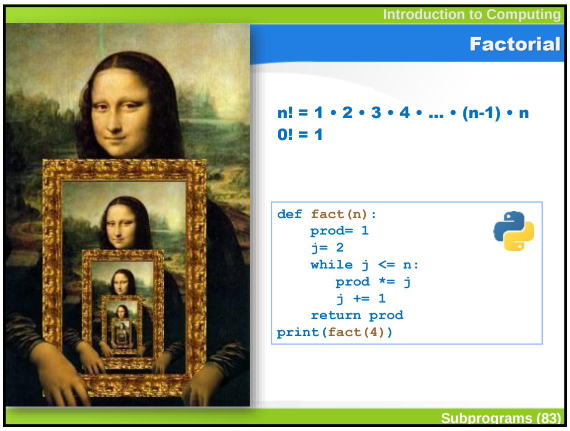


Introduction to Computing

Factorial

n! = 1 • 2 • 3 • 4 • ... • (n-1) • n
4! = 1 • 2 • 3 • 4 = 24

Subprograms (82)



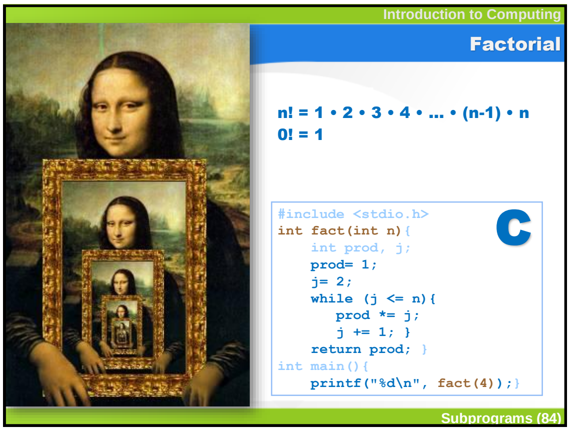
Introduction to Computing

Factorial

n! = 1 • 2 • 3 • 4 • ... • (n-1) • n
0! = 1

```
def fact(n):
    prod= 1
    j= 2
    while j <= n:
       prod *= j
       j += 1
    return prod
print(fact(4))
```

Subprograms (83)



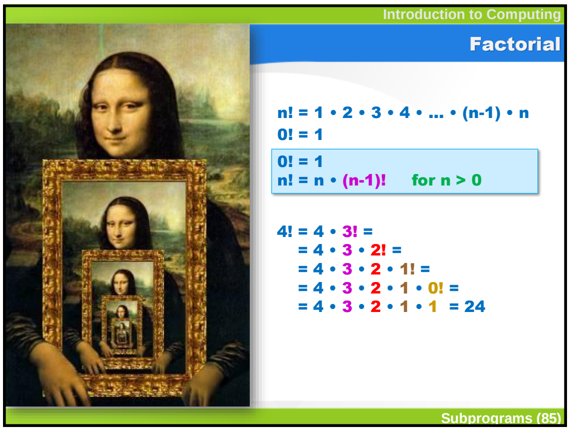
Introduction to Computing

Factorial

n! = 1 • 2 • 3 • 4 • ... • (n-1) • n
0! = 1

```
#include <stdio.h>
int fact(int n){
    int prod, j;
    prod= 1;
    j= 2;
    while (j <= n){
       prod *= j;
       j += 1; }
    return prod; }
int main(){
    printf("%d\n", fact(4));}
```

Subprograms (84)



Introduction to Computing

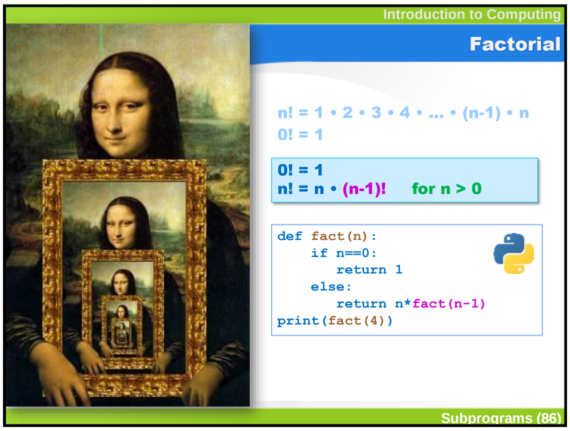
Factorial

n! = 1 • 2 • 3 • 4 • ... • (n-1) • n
0! = 1

0! = 1
n! = n • (n-1)!   for n > 0

4! = 4 • 3! =
 = 4 • 3 • 2! =
 = 4 • 3 • 2 • 1! =
 = 4 • 3 • 2 • 1 • 0! =
 = 4 • 3 • 2 • 1 • 1 = 24

Subprograms (85)



Introduction to Computing

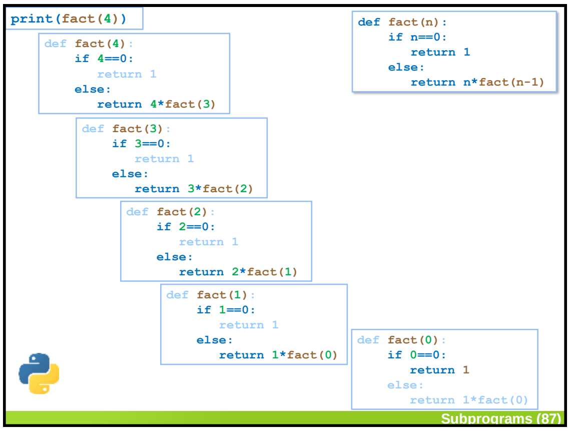
Factorial

n! = 1 • 2 • 3 • 4 • ... • (n-1) • n
0! = 1

0! = 1
n! = n • (n-1)!   for n > 0

```
def fact(n):
    if n==0:
       return 1
    else:
       return n*fact(n-1)
print(fact(4))
```

Subprograms (86)



```
print(fact(4))
```

```
def fact(4):
    if 4==0:
       return 1
    else:
       return 4*fact(3)
```

```
def fact(3):
    if 3==0:
       return 1
    else:
       return 3*fact(2)
```

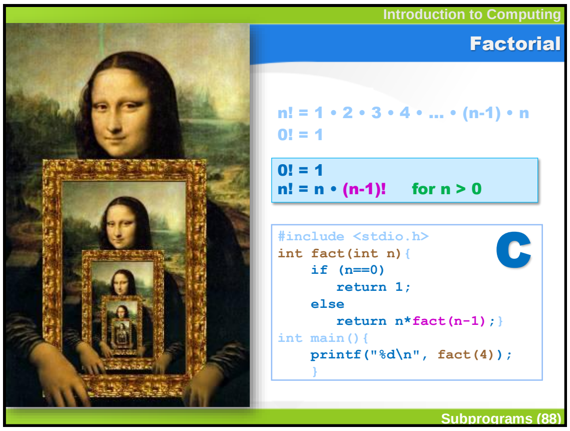
```
def fact(2):
    if 2==0:
       return 1
    else:
       return 2*fact(1)
```

```
def fact(1):
    if 1==0:
       return 1
    else:
       return 1*fact(0)
```

```
def fact(0):
    if 0==0:
       return 1
    else:
       return 1*fact(0)
```

```
def fact(n):
    if n==0:
       return 1
    else:
       return n*fact(n-1)
```

Subprograms (87)



Introduction to Computing

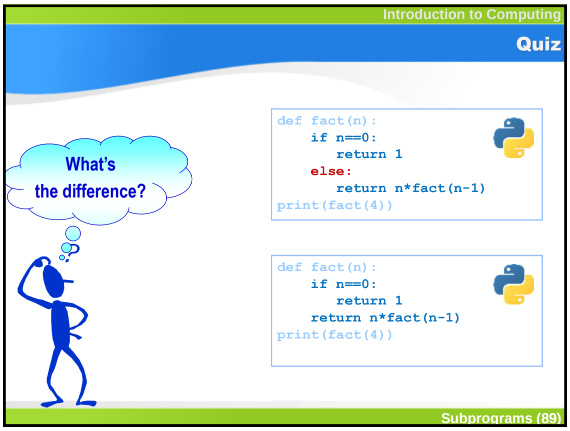
Factorial

n! = 1 • 2 • 3 • 4 • ... • (n-1) • n
0! = 1

0! = 1
n! = n • (n-1)!   for n > 0

```
#include <stdio.h>
int fact(int n){
    if (n==0)
       return 1;
    else
       return n*fact(n-1);}
int main(){
    printf("%d\n", fact(4));
  }
```

Subprograms (88)



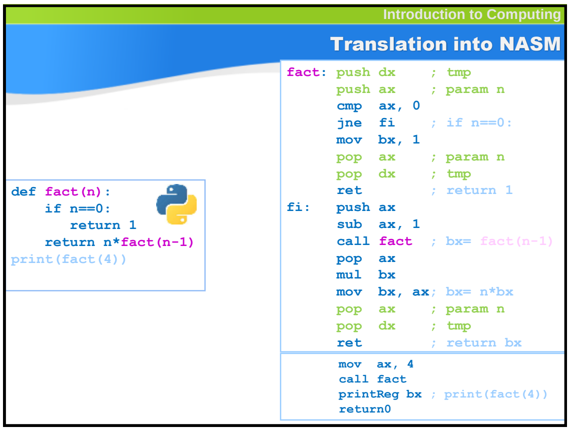
Introduction to Computing

Quiz

```
def fact(n):
    if n==0:
       return 1
    else:
       return n*fact(n-1)
print(fact(4))
```

```
def fact(n):
    if n==0:
       return 1
    return n*fact(n-1)
print(fact(4))
```

Subprograms (89)



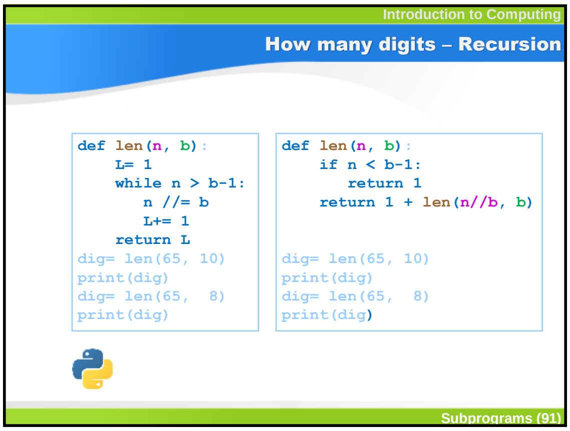
Introduction to Computing

Translation into NASM

```
def fact(n):
    if n==0:
       return 1
    return n*fact(n-1)
print(fact(4))
```

```
fact: push dx     ; tmp
      push ax     ; param n
      cmp  ax, 0
      jne  fi     ; if n==0:
      mov  bx, 1
      pop  ax     ; param n
      pop  dx     ; tmp
      ret         ; return 1
fi:   push ax
      sub  ax, 1
      call fact   ; bx= fact(n-1)
      pop  ax
      mul  bx
      mov  bx, ax; bx= n*bx
      pop  ax     ; param n
      pop  dx     ; tmp
      ret         ; return bx
```

```
      mov  ax, 4
      call fact
      printReg bx ; print(fact(4))
      return0
```



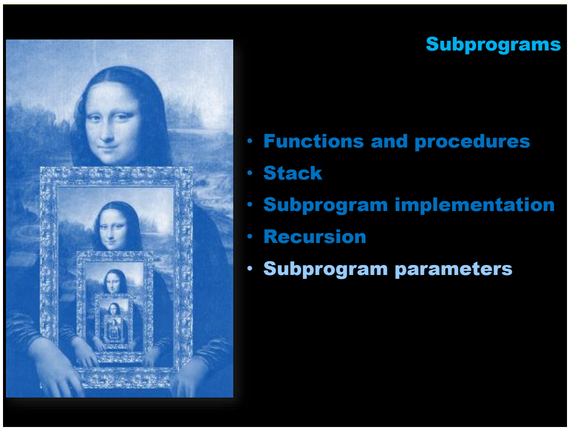
Introduction to Computing

How many digits – Recursion

```
def len(n, b):
    L= 1
    while n > b-1:
        n //= b
        L+= 1
    return L
dig= len(65, 10)
print(dig)
dig= len(65,  8)
print(dig)
```

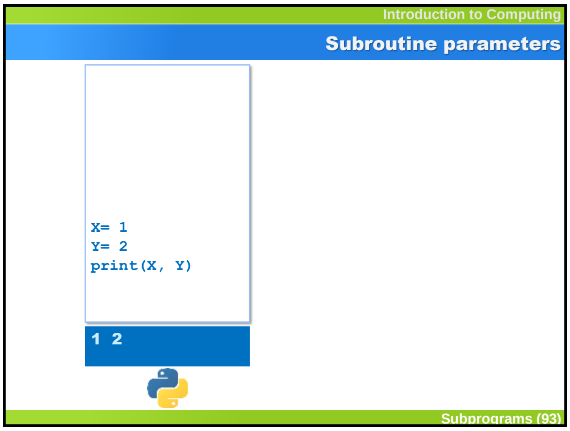
```
def len(n, b):
    if n < b-1:
        return 1
    return 1 + len(n//b, b)


dig= len(65, 10)
print(dig)
dig= len(65,  8)
print(dig)
```

Subprograms (91)



Subprograms

- Functions and procedures
- Stack
- Subprogram implementation
- Recursion
- Subprogram parameters



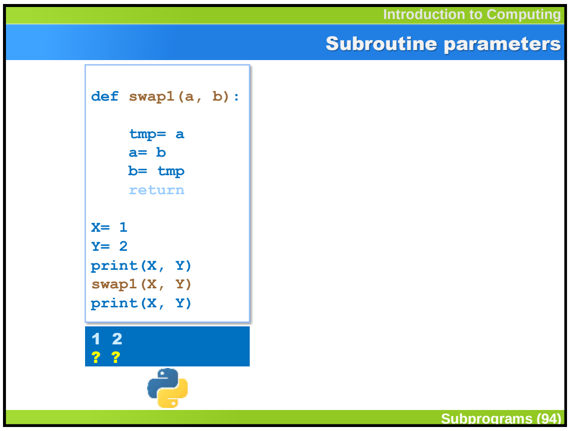
Introduction to Computing

Subroutine parameters

```
X= 1
Y= 2
print(X, Y)
```

```
1 2
```

Subprograms (93)



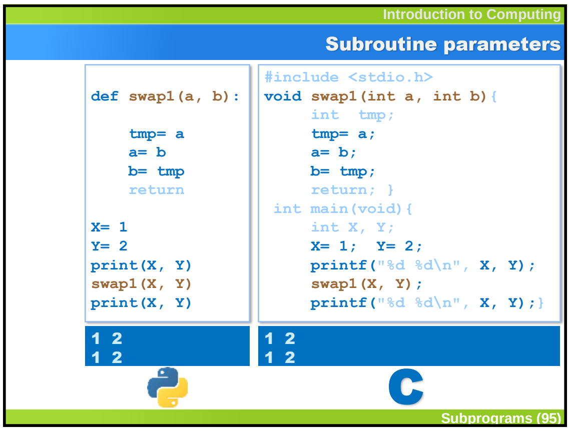
Introduction to Computing

Subroutine parameters

```
def swap1(a, b):

    tmp= a
    a= b
    b= tmp
    return

X= 1
Y= 2
print(X, Y)
swap1(X, Y)
print(X, Y)
```

```
1 2
? ?
```

Subprograms (94)



Introduction to Computing

Subroutine parameters

```
def swap1(a, b):

    tmp= a
    a= b
    b= tmp
    return

X= 1
Y= 2
print(X, Y)
swap1(X, Y)
print(X, Y)
```

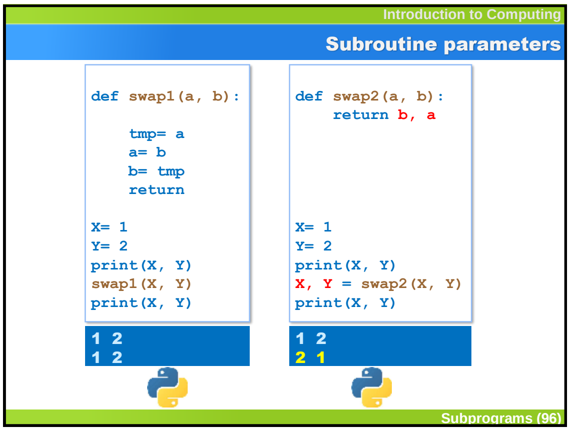
```
1 2
1 2
```

```
#include <stdio.h>
void swap1(int a, int b){
     int  tmp;
     tmp= a;
     a= b;
     b= tmp;
     return; }
 int main(void){
     int X, Y;
     X= 1;  Y= 2;
     printf("%d %d\n", X, Y);
     swap1(X, Y);
     printf("%d %d\n", X, Y);}
```

```
1 2
1 2
```

C

Subprograms (95)



Introduction to Computing

Subroutine parameters

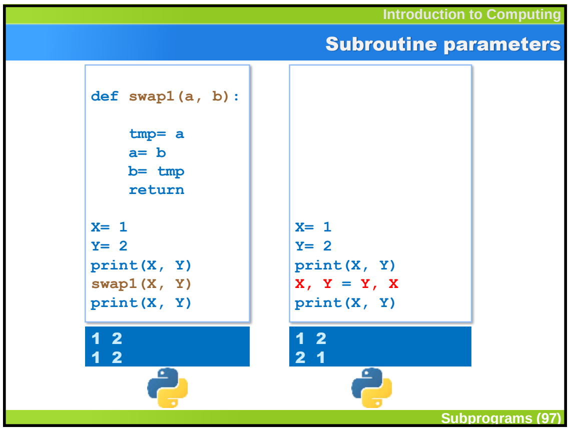
```
def swap1(a, b):

    tmp= a
    a= b
    b= tmp
    return

X= 1
Y= 2
print(X, Y)
swap1(X, Y)
print(X, Y)
```

```
1 2
1 2
```

```
def swap2(a, b):
    return b, a






X= 1
Y= 2
print(X, Y)
X, Y = swap2(X, Y)
print(X, Y)
```

```
1 2
2 1
```

Subprograms (96)



Introduction to Computing

Subroutine parameters

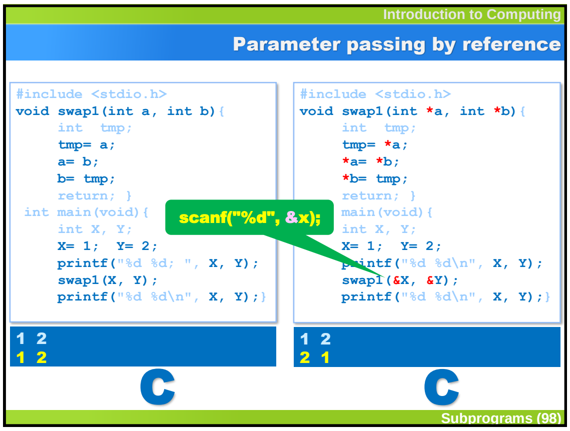
```
def swap1(a, b):

    tmp= a
    a= b
    b= tmp
    return

X= 1
Y= 2
print(X, Y)
swap1(X, Y)
print(X, Y)
```

```
1 2
1 2
```

```
X= 1
Y= 2
print(X, Y)
X, Y = Y, X
print(X, Y)
```

```
1 2
2 1
```

Subprograms (97)



Introduction to Computing

Parameter passing by reference

```
#include <stdio.h>
void swap1(int a, int b){
     int  tmp;
     tmp= a;
     a= b;
     b= tmp;
     return; }
 int main(void){
     int X, Y;
     X= 1;  Y= 2;
     printf("%d %d; ", X, Y);
     swap1(X, Y);
     printf("%d %d\n", X, Y);}
```

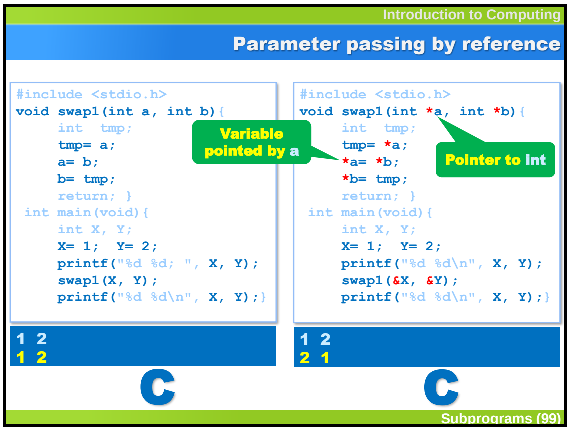
```
1 2
1 2
```

scanf("%d", &x);

```
#include <stdio.h>
void swap1(int *a, int *b){
     int  tmp;
     tmp= *a;
     *a= *b;
     *b= tmp;
     return; }
     main(void){
     int X, Y;
     X= 1;  Y= 2;
     printf("%d %d\n", X, Y);
     swap1(&X, &Y);
     printf("%d %d\n", X, Y);}
```

```
1 2
2 1
```

Subprograms (98)



Introduction to Computing

Parameter passing by reference

```
#include <stdio.h>
void swap1(int a, int b){
     int  tmp;
     tmp= a;
     a= b;
     b= tmp;
     return; }
 int main(void){
     int X, Y;
     X= 1;  Y= 2;
     printf("%d %d; ", X, Y);
     swap1(X, Y);
     printf("%d %d\n", X, Y);}
```

```
1 2
1 2
```

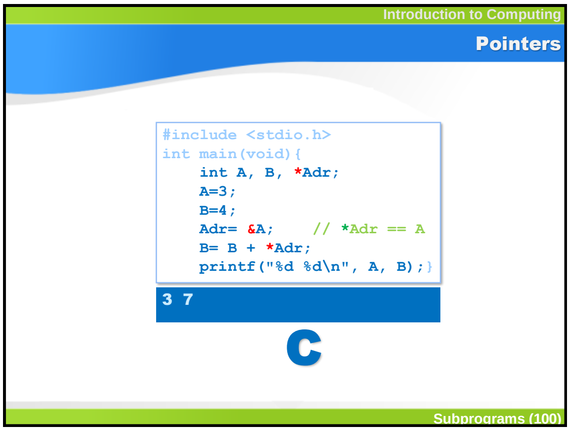
Variable pointed by a

Pointer to int

```
#include <stdio.h>
void swap1(int *a, int *b){
     int  tmp;
     tmp= *a;
     *a= *b;
     *b= tmp;
     return; }
 int main(void){
     int X, Y;
     X= 1;  Y= 2;
     printf("%d %d\n", X, Y);
     swap1(&X, &Y);
     printf("%d %d\n", X, Y);}
```

```
1 2
2 1
```

Subprograms (99)



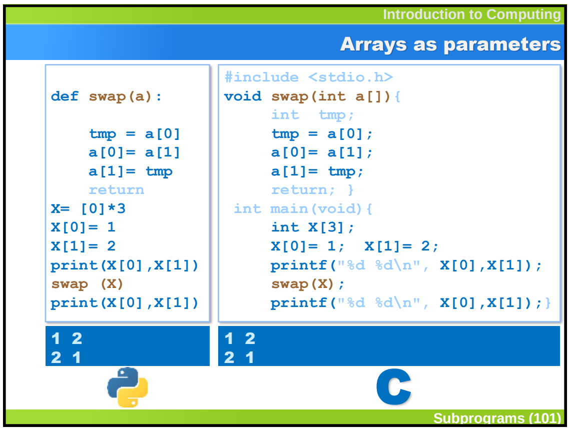
Introduction to Computing

Pointers

```
#include <stdio.h>
int main(void){
     int A, B, *Adr;
     A=3;
     B=4;
     Adr= &A;     // *Adr == A
     B= B + *Adr;
     printf("%d %d\n", A, B);}
```

```
3 7
```

Subprograms (100)



Introduction to Computing

Arrays as parameters

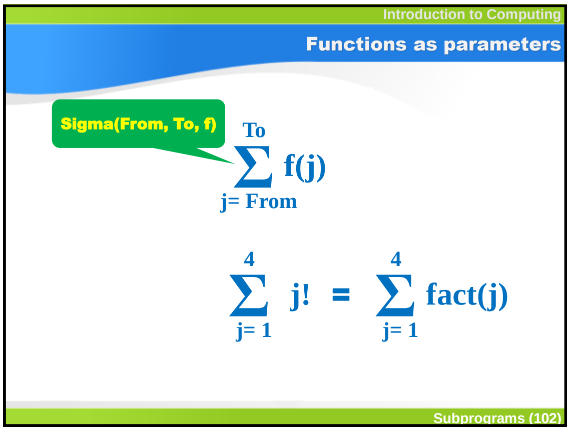
```
def swap(a):

    tmp = a[0]
    a[0]= a[1]
    a[1]= tmp
    return
X= [0]*3
X[0]= 1
X[1]= 2
print(X[0],X[1])
swap (X)
print(X[0],X[1])
```

```
1 2
2 1
```

```
#include <stdio.h>
void swap(int a[]){
     int  tmp;
     tmp = a[0];
     a[0]= a[1];
     a[1]= tmp;
     return; }
 int main(void){
     int X[3];
     X[0]= 1;  X[1]= 2;
     printf("%d %d\n", X[0],X[1]);
     swap(X);
     printf("%d %d\n", X[0],X[1]);}
```

```
1 2
2 1
```

Subprograms (101)



Introduction to Computing

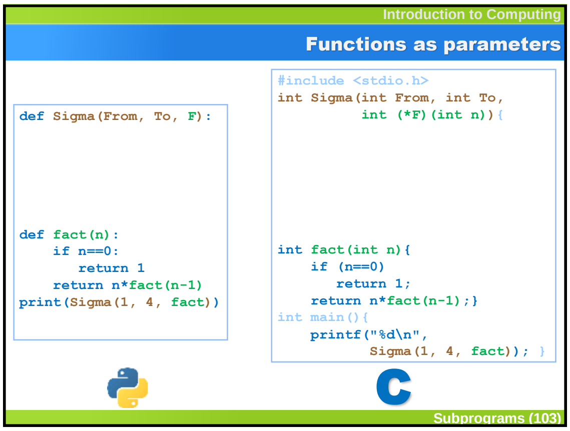
Functions as parameters

Sigma(From, To, f)

$$\sum_{j=From}^{To} f(j)$$

$$\sum_{j=1}^{4} j! = \sum_{j=1}^{4} fact(j)$$

Subprograms (102)



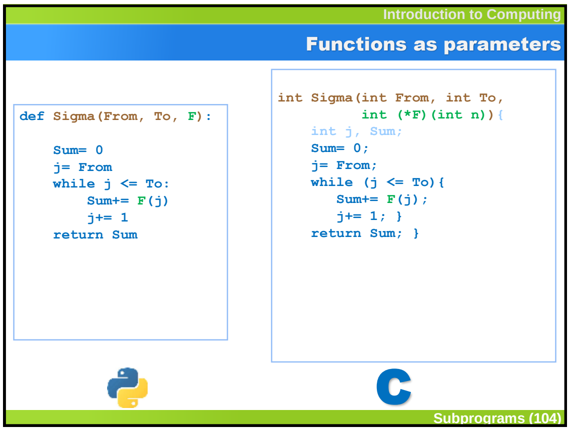
Introduction to Computing

Functions as parameters

```
def Sigma(From, To, F):




def fact(n):
    if n==0:
       return 1
    return n*fact(n-1)
print(Sigma(1, 4, fact))
```

```
#include <stdio.h>
int Sigma(int From, int To,
          int (*F)(int n)){




int fact(int n){
    if (n==0)
       return 1;
    return n*fact(n-1);}
int main(){
    printf("%d\n",
           Sigma(1, 4, fact)); }
```

Subprograms (103)



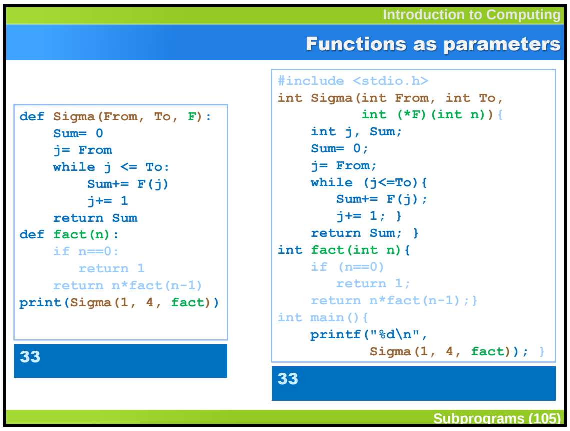
Introduction to Computing

Functions as parameters

```
def Sigma(From, To, F):

    Sum= 0
    j= From
    while j <= To:
        Sum+= F(j)
        j+= 1
    return Sum
```

```
int Sigma(int From, int To,
          int (*F)(int n)){
    int j, Sum;
    Sum= 0;
    j= From;
    while (j <= To){
       Sum+= F(j);
       j+= 1; }
    return Sum; }
```

Subprograms (104)



Introduction to Computing

Functions as parameters

```
def Sigma(From, To, F):
    Sum= 0
    j= From
    while j <= To:
        Sum+= F(j)
        j+= 1
    return Sum
def fact(n):
    if n==0:
       return 1
    return n*fact(n-1)
print(Sigma(1, 4, fact))
```

```
33
```

```
#include <stdio.h>
int Sigma(int From, int To,
          int (*F)(int n)){
    int j, Sum;
    Sum= 0;
    j= From;
    while (j<=To){
       Sum+= F(j);
       j+= 1; }
    return Sum; }
int fact(int n){
    if (n==0)
       return 1;
    return n*fact(n-1);}
int main(){
    printf("%d\n",
           Sigma(1, 4, fact)); }
```

```
33
```

Subprograms (105)